Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume un

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

Il ne s'agit de rien d'autre qu'une reprise de la série Programmer en Python, parties 1 à 8, numéros 27 à 34 ; pas de chichi, juste les faits.

Gardez à l'esprit la date de publication ; les versions actuelles du matériel et des logiciels peuvent être différentes de celles illustrées. Il vous est recommandé de bien vérifier la version de votre matériel et des logiciels avant d'essayer d'émuler les tutoriels dans ces numéros spéciaux. Il se peut que vous ayez des logiciels plus récents ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

Site Web :
http://www.fullcirclemagazine.org/

Forums :
http://ubuntuforums.org/forumdisplay.php?f=270

IRC : #fullcirclemagazine on chat.freenode.net

Équipe Full Circle

Rédacteur en chef : Ronnie Tucker
(aka: RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia
(aka: admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcast : Robin Catling
(aka RobinCatling)
podcast@fullcirclemagazine.org
Dir. comm. : Robert Clipsham
(aka: mrmonday) -
mrmonday@fullcirclemagazine.org

TUTORIEL

Programmer en Python - Partie 1

Greg Walters

Parmi tous les langages de programmation disponibles à l'heure actuelle, Python est un des plus faciles à apprendre. Python a été créé à la fin des années 80 et a énormément mûri depuis. Il est pré-installé dans la plupart des distributions Linux et est souvent un des plus négligés quand on choisit un langage à apprendre. Nous allons nous confronter à la programmation en ligne de commande dans cet article. Dans un prochain article, nous jouerons avec la programmation d'une interface graphique (Graphical User Interface = GUI). Sautons à l'eau en créant une application simple.

Notre premier programme

Écrivons quelques lignes de code en utilisant un éditeur tel que gedit. Ensuite seulement nous observerons le rôle de chaque ligne puis nous poursuivrons.

Saisissez les 4 lignes suivantes :

```
#!/usr/bin/env python

print 'Bonjour, je suis un
programme ecrit en Python.'

nom = raw_input("Quel est
votre nom ? ")

print "Salut, " + nom + "!"
```

C'est tout ce qu'il y a à faire. Enregistrez le fichier sous le nom « hello.py » où vous voudrez. Je vous suggère votre dossier personnel, dans un dossier appelé « python_exemples ». Cet exemple simple montre à quel point il est aisé de programmer en Python. Avant de pouvoir exécuter le programme, nous devons le rendre exécutable.

Saisissez :

```
chmod +x hello.py
```

dans le dossier où est enregistré le fichier source. À présent, exécutons le programme.

```
greg@earth:~/python_exemples$
./hello.py

Bonjour, je suis un
programme ecrit en Python.

Quel est votre nom ? Ferd
Burphel

Salut, Ferd Burphel!

greg@earth:~/python_exemples$
```

C'était simple. Maintenant détaillons chaque ligne du programme.

```
#!/usr/bin/env python
```

Cette ligne dit au système que c'est un programme Python et qu'il faut utiliser l'interpréteur par défaut pour exécuter le programme.

```
print 'Bonjour, je suis un
programme ecrit en Python.'
```

Écrit tel quel, ceci affiche la première ligne « Bonjour, je suis un programme ecrit en Python. » dans le terminal :

```
nom = raw_input("Quel est
votre nom ? ")
```

Celle-ci est un peu plus complexe. Cette ligne a deux parties: « nom = » ainsi que « raw_input("Quel est votre nom ? ") ». Regardons d'abord la seconde partie. La commande raw_input va poser la question dans le terminal ("Quel est votre nom ? "), et ensuite va attendre que l'utilisateur (vous) écrive quelque chose (suivi de {Entrée}).

Maintenant, regardons la première partie : nom =. Cette partie de la commande attribue une valeur à une variable appelée « nom ». Qu'est-ce qu'une variable ? Imaginez une boîte à chaussures. Vous pouvez utiliser cette boîte pour ranger toutes sortes de choses - des chaussures, des composants d'ordinateur, des papiers, n'importe quoi. Peu importe ce qu'il y a dedans, c'est simplement entreposé là. Dans notre cas, elle contient ce que vous écrivez. En ce qui me concerne, j'ai écrit « Ferd Burphel ». Python, dans cet exemple, prend simplement ce que vous avez tapé et le range dans la boîte « nom » pour pouvoir l'utiliser plus tard dans le programme.

```
print "Salut, " + nom + "!"
```

Encore une fois, nous utilisons la commande « print » pour afficher quelque chose sur l'écran – dans notre cas : « Salut, », plus ce qui se trouve dans la variable « nom » puis un point d'exclamation à la fin. Ici nous concaténons, ou collons ensemble, trois morceaux d'information : « Salut, », les données de la variable « nom » et un point d'exclamation.

Maintenant prenons un instant pour examiner les choses un peu plus en profondeur avant de travailler sur l'exemple suivant. Ouvrez un terminal et saisissez :

```
python
```

Vous devriez obtenir chose comme ceci :

```
greg@earth:~/python_exemples$
python

Python 2.5.2 (r252:60911,
Oct 5 2008, 19:24:49)

[GCC 4.3.2] on linux2

Type "help", "copyright",
"credits" or "license" for
more information.

>>>
```

Vous êtes maintenant dans le shell Python. Là, vous pouvez faire beaucoup de choses, mais examinons ce qui vient de nous être présenté avant de continuer. La première chose que vous avez sûrement remarqué est la version de python - la mienne étant 2.5.2. Ensuite, vous avez lu une phrase indiquant que, pour voir l'aide du programme, vous devez écrire « help » après le prompt. Je vous laisserai faire ça vous même. Maintenant, saisissez :

```
print 2+2
```

et appuyez sur « Entrée ». Vous devriez obtenir en retour

```
>>> print 2+2
4
>>>
```

Remarquez que l'on a tapé le mot « print » en minuscule. Que se passerait-il si nous tapions « Print 2+2 » ? La réponse de l'interpréteur est la suivante :

```
>>> Print 2+2
    File "<stdin>", line 1
        Print 2+2
              ^
SyntaxError: invalid syntax
>>>
```

C'est parce que le mot « print » est une commande connue, tandis que « Print » n'en est pas une. La casse est très importante en Python.

Maintenant jouons un peu plus avec les variables. Saisissez :

```
var = 2+2
```

Vous verrez que rien ne se passe, excepté que Python renvoie le prompt « >>> ». Tout va bien. Ce que nous avons dit de faire à Python est de créer une variable (une boîte) appelée « var », et de mettre dedans le résultat de la somme de « 2+2 ». Pour voir à présent ce que contient « var », saisissez :

```
print var
```

et appuyez sur Entrée.

```
>>> print var
4
>>>
```

Maintenant nous pouvons réutiliser « var » autant de fois que nous le désirons en lieu et place du nombre « 4 », comme ceci :

```
>>> print var * 2
>>> print var
4
>>>
```

« var » n'a pas varié. Elle contient toujours la somme de 2+2, soit 4.

C'est ici, bien évidemment, un tutoriel de programmation simple pour débutants. La complexité de nos exemples va croître dans les numéros à venir. Mais pour l'heure, regardons quelques exemples de variables.

Dans l'interpréteur, saisissez :

```
>>> chaine = 'Le temps est
venu pour tout honnête
homme de venir en aide au
parti !'

>>> print chaine

Le temps est venu pour tout
honnête homme de venir en
aide au parti !

>>>
```

Nous avons créé une variable nommée « chaine » contenant la valeur 'Le temps est venu pour tout honnête homme de venir en aide au parti !'. Dès lors (et aussi longtemps que nous ne quittons pas cette instance de l'interpréteur), notre variable « chaine » restera inchangée à moins que nous voulions la changer. Que se passerait-il si nous voulions multiplier cette variable par 4 ?

```
>>> print chaine * 4

Le temps est venu pour tout
honnête homme de venir en
aide au parti !Le temps est
venu pour tout honnête homme
de venir en aide au
parti !Le temps est venu
pour tout honnête homme de
venir en aide au parti !Le
temps est venu pour tout
honnête homme de venir en
aide au parti !

>>>
```

Ce n'est pas exactement ce à quoi vous vous attendiez, n'est-ce pas ? La valeur de « chaine » a été écrite 4 fois. Pourquoi ? Et bien l'interpréteur sait que « chaine » est une chaîne de caractères et pas un nombre. Vous ne pouvez pas faire des opérations mathématiques avec des chaînes de caractères.

Que se passerait-il si nous avions « une variable appelée « s » qui contiendrait « 4 », comme ceci :

```
>>> s = '4'
>>> print s
4
```

On dirait que « s » contient le nombre entier 4, mais ce n'est pas le cas. À la place, il contient le nombre 4 en tant que chaîne de caractères. C'est pourquoi, si on écrit « print s * 4 » on obtient :

```
>>> print s*4
4444
>>>
```

Encore une fois, l'interpréteur sait que « s » est une chaîne de caractères, et non une valeur numérique. Il le sait parce que nous avons entouré le nombre 4 de guillemets simples, le transformant ainsi en chaîne de caractères.

On peut prouver qu'il s'agit bien d'une chaîne en saisissant « print type(s) » pour voir de quel type le système pense que cette variable est.

```
>>> print type(s)
<type 'str'>
>>>
```

Confirmation. Elle est du type « str » c'est-à-dire : chaîne de caractères. Si nous voulions l'utiliser en tant que valeur numérique, nous pourrions faire comme suit :

```
>>> print int(s) * 4
16
>>>
```

La chaîne (s), qui contient « 4 », a maintenant été convertie en un entier multiplié par 4, pour donner 16.

Maintenant vous connaissez les commandes « print », « raw_input », l'assignation des variables, et la différence entre les chaînes et les entiers.

Allons maintenant un peu plus loin. Dans l'interpréteur Python, saisissez « quit() » pour revenir au terminal système.

Une boucle « for » facile

Maintenant, voyons la programmation d'une boucle simple. Revenez à l'éditeur de texte et saisissez le programme suivant :

```
#! /usr/bin/env python

for cmpt in range(0,10):

    print cmpt
```

N'oubliez surtout pas d'insérer une tabulation avant « print cmpt ».

C'est important. Python n'utilise pas les parenthèses « (» ni les accolades « { » comme le font les autres langages pour séparer les différents blocs de code. Il utilise, à la place, l'indentation.

Enregistrez le programme sous le nom « for_loop.py ». Avant d'essayer de l'exécuter, parlons un peu de cette boucle « for ».

Une boucle, c'est du code qui exécute une instruction particulière, ou un ensemble d'instructions, un certain nombre de fois. Dans notre programme, nous bouclons 10 fois, en affichant la valeur de la variable « cmpt » (abréviation de « compteur »). En clair, la commande est « assigne la valeur 0 à la variable cmpt, boucle 10 fois en imprimant la valeur de cette variable, ajoute 1 à cmpt et recommence ». Ça semble assez simple.

Le bout de code « range(0,10) » dit de commencer à 0, de boucler jusqu'à ce que la valeur de cmpt soit égale à 10 et de quitter.

Maintenant, comme vu précédemment, faites un :

```
chmod +x for_loop.py
```

et exécutez le programme avec :

```
./for_loop.py
```

dans un terminal.

```
greg@earth:~/python_exemples$
./for_loop.py
0
1
2
3
4
5
6
7
8
9
greg@earth:~/python_exemples$
```

Bon, ça a l'air de fonctionner, mais pourquoi est-ce qu'il compte seulement jusqu'à 9. Regardez encore une fois le résultat. Il y a bien 10 nombres affichés, commençant à 0 et se terminant à 9. C'est précisément ce que nous lui avions demandé de faire -

afficher la valeur de cmpt 10 fois, en incrémentant la variable de 1 à chaque itération et quitter aussitôt que la valeur égale 10.

Maintenant vous comprenez que, si programmer peut être simple, ça peut être complexe également, et vous devez être sûr de ce que vous demandez au système. Si vous changez la définition de la zone à couvrir pour « range(1,10) », il commencera à compter à 1, mais finira toujours à 9, puisque la boucle est quittée aussitôt que cmpt égale 10. Ainsi, pour demander l'affichage de « 1,2,3,4,5,6,7,8,9,10 », nous devons utiliser « range(1,11) » - puisque la boucle « for » est quittée aussitôt que la limite supérieure de l'intervalle est atteinte.

Notez également la syntaxe de l'instruction. C'est « for variable in range (valeur de départ, valeur limite): ». Les « : » annoncent à l'interpréteur qu'il doit s'attendre à ce qu'un bloc de code indenté commence en dessous. Il est très important de ne pas oublier les « : » et d'indenter le code jusqu'à ce que le bloc soit terminé.

Si nous modifions notre programme comme suit :

```
#! /usr/bin/env python

for cmpt in range(1,11):

      print cmpt

print 'Fini'
```

Nous aurons un résultat ressemblant à ceci :

```
greg@earth:~/python_exemples$
./for_loop.py
1
2
3
4
5
6
7
8
9
10
Fini
greg@earth:~/python_exemples$
```

Assurez-vous que l'indentation est correcte. Souvenez-vous, l'indentation signale le formatage du bloc. Nous nous occuperons davantage de l'idée d'indentation dans notre prochain tutoriel.

C'est à peu près tout pour cette fois-ci. La prochaine fois nous ferons un rappel et irons plus loin, avec plus d'instructions Python. En attendant, vous aimeriez peut-être installer un éditeur spécial pour Python, comme Dr. Python, ou SPE (Stani's Python Editor). Les deux sont disponibles dans Synaptic.

TUTORIEL

Greg Walters

Programmer en Python - Partie 2

Correctif à la partie 1

J'ai reçu un courriel de David Turner qui me suggère que l'utilisation de la touche Tab pour l'indentation du code est quelque peu trompeuse car il se peut que certains éditeurs utilisent plus ou moins de quatre espaces par décalage. Ceci est exact. Beaucoup de programmeurs Python (dont moi- même) gagnons du temps en paramétrant dans l'éditeur la touche T à quatre espaces. Cependant le problème est qu'il est possible que l'éditeur de quelqu'un d'autre n'ait pas la même configuration que la vôtre, ce qui peut rendre votre code laid et mener à d'autres problèmes. En conclusion, prenez l'habitude d'utiliser des espaces au lieu de la touche Tab.

Dans le dernier épisode, nous avons vu un programme élémentaire qui utilisait raw_input pour obtenir une réponse de l'utilisateur, des types de variables simples et une boucle élémentaire utilisant l'instruction « for ». Dans cette partie, nous approfondirons la notion de variables et écrirons un peu plus de programmes.

LISTES

Intéressons-nous à un autre type de variables appelées listes. Dans les autres langages, une liste serait considérée comme un tableau. En reprenant l'analogie de la boîte à chaussures, un tableau (ou une liste) serait un certain nombre de boîtes collées côtes à côtes contenant des choses. Par exemple, on pourrait mettre des fourchettes dans la première, des couteaux dans une autre et des cuillères dans une autre. Observons une liste simple. Une liste facile à imaginer serait celle des noms des mois. Nous pourrions la coder comme cela :

```
mois =
['Jan','Fév','Mars','Avr','Mai','
Juin','Juil','Août','Sept','Oct',
'Nov','Déc']
```

Pour créer la liste, nous entourons toutes les valeurs avec des crochets (« [» et «] »). La liste s'appelle « mois ». Pour l'utiliser, nous pourrions écrire quelque chose comme « print mois[0] » ou « mois[1] » (ce qui afficherait « Jan » ou « Fév »). Souvenez-vous que nous commençons toujours à compter par zéro. Pour trouver la longueur de la liste, nous pouvons utiliser :

```
print len(mois)
```

qui retourne 12.

Un autre exemple de liste pourrait être celles des rubriques d'un livre de cuisine. Par exemple :

```
rubriques = ['Plats
principaux','Viande','Poisson','
Soupe','Gâteaux']
```

rubriques[0] serait alors « Plats principaux » et rubriques[4] serait « Gâteaux ». Toujours très simple.

Je suis sûr que vous pensez déjà à toutes les choses que vous pourriez faire avec une liste.

Pour l'instant, nous avons créé une liste contenant des chaînes de caractères. Vous pouvez également créer une liste contenant des entiers. Reprenons notre liste des mois, nous pourrions créer une liste contenant le nombre de jours pour chacun d'eux :

```
JoursDansMois =
[31,28,31,30,31,30,31,31,30,31,30
,31]
```

Si on saisissait « print JoursDansMois[1] » (pour février), on obtiendrait 28 qui est un entier. Remarquez que j'ai appelé la liste JoursDansMois. De manière plus simple, j'aurais pu utiliser « joursdansmois » ou juste « X »... mais ce n'est pas aussi facile à lire. Les bonnes pratiques de la programmation suggèrent (ce qui est sujet à interprétation) que les noms des variables soient faciles à comprendre. Nous verrons pourquoi plus tard. Nous allons nous amuser un peu plus avec les listes dans un instant.

Avant de passer à notre prochain exemple de programme, intéressons-nous à quelques autres détails à propos de Python.

Suppléments sur les chaînes

Nous avons brièvement discuté des chaînes dans la partie 1. Intéressons-nous aux chaînes d'un peu plus près. Une chaîne est une série de caractères. Rien de plus. En fait, nous pouvons voir les chaînes comme un tableau de caractères. Par exemple, si nous assignons la chaîne « Le moment est venu de » à une variable nommée

« phrase » et si nous voulons savoir quelle est la deuxième lettre, nous pouvons saisir :

```
phrase = 'Le moment est venu de'
print phrase[1]
```

Le résultat serait « e ». Souvenez-vous, nous comptons toujours à partir de 0 donc la première lettre serait [0], la seconde [1], la troisième [2], etc. Si nous voulons trouver les lettres en commençant de la position 3 jusqu'à la position 9, nous ferions :

```
print phrase[3:9]
```

qui retourne « moment ». Comme pour notre boucle « for » de la partie 1, le comptage s'arrête à 9 mais ne retourne pas le neuvième caractère qui serait l'espace après « moment ».

Nous pouvons obtenir la longueur de notre chaîne en utilisant la fonction len() :

```
print len(phrase)
```

qui retourne 21. Si nous voulons trouver la position du mot « moment » dans la chaîne, nous pouvons saisir :

```
pos = phrase.find('moment')
```

Maintenant la variable « pos » (abréviation pour position) contient 3, ce qui veut dire que « moment » commence à la position 3 de notre chaîne. Si nous demandons à la commande « find » de trouver un mot ou une phrase qui n'existe pas dans la chaîne comme ici :

```
pos = phrase.find('pommes')
```

la valeur enregistrée dans « pos » serait -1.

Nous pouvons également récupérer chaque mot distinct de la chaîne en utilisant la commande split. Nous allons diviser (ou casser) la chaîne à chaque caractère « espace » en utilisant :

```
print phrase.split(' ')
```

qui retourne une liste contenant ['Le', 'moment', 'est', 'venu', 'de']. C'est vraiment un truc très puissant. Il y a plein d'autres fonctions intégrées pour les chaînes ; nous les utiliserons plus tard.

Substitution littérale

Il y a encore une autre chose que je voudrais présenter avant de commencer notre prochain exemple de programmation. Lorsque nous voulons afficher quelque chose qui contient du texte littéral aussi bien que du texte contenu dans une variable, nous pouvons utiliser ce qui s'appelle la substitution des variables. Pour faire cela, c'est très simple. Si nous voulons remplacer une chaîne, nous utilisons « %s » puis nous disons à Python par quoi il faut la remplacer. Par exemple, pour afficher un mois de notre liste précédente, nous pouvons utiliser :

```
print 'Mois = %s' % mois[0]
```

Cela affiche « Mois = Jan ». Si nous voulons remplacer un entier, nous utilisons « %d ». Observez l'exemple ci-dessous :

```
Mois =
['Jan','Fév','Mars','Avr','Mai','
Juin','Juil','Août','Sept','Oct',
'Nov','Déc']
JoursDansMois =
[31,28,31,30,31,30,31,31,30,31,30
,31]
for cmptr in range(0,12):
    print '%s a %d jours.' %
(Mois[cmptr],JoursDansMois[cmptr])
```

Le résultat de ce code est :

```
Jan a 31 jours.
Fév a 28 jours.
Mars a 31 jours.
Avr a 30 jours.
Mai a 31 jours.
Juin a 30 jours.
Juil a 31 jours.
Août a 31 jours.
Sept a 30 jours.
Oct a 31 jours.
Nov a 30 jours.
Déc a 31 jours
```

Une chose importante à comprendre maintenant est l'utilisation des apostrophes simples et doubles. Si vous affectez une variable à une chaîne de cette manière :

```
ch = 'Le moment est venu de'
```

ou comme ceci :

```
ch = "Le moment est venu de"
```

le résultat est le même. Cependant, si vous devez inclure une apostrophe simple dans la chaîne comme ceci :

```
ch = 'Il dit qu'il arrive'
```

il y aura une erreur de syntaxe. Vous devez l'assigner de cette manière :

```
ch = "Il dit qu'il arrive"
```

Considérez-le ainsi : pour définir une chaîne, vous devez l'entourer par un type d'apostrophes - une au début, l'autre à la fin - et elles doivent correspondre. Si vous devez mélanger des apostrophes, utilisez pour celles de l'extérieur, celles qui ne se trouvent pas dans la chaîne comme ci-dessus. Vous vous dites peut-être : que faire si je dois définir une chaîne comme « Elle a dit “Ne t'inquiète pas” » ?

Dans ce cas, vous devez faire comme cela :

```
st = 'Elle a dit "Ne
t\'inquiète pas"'
```

Remarquez la barre oblique inversée (anti-slash) devant l'apostrophe simple de « Ne t'inquiète pas ». Ceci est appelé un caractère d'échappement et il indique à Python d'afficher (dans ce cas) l'apostrophe simple et de ne pas la traiter comme un délimiteur de chaînes. D'autres séquences avec caractère d'échappement (pour n'en citer que quelques-unes) seraient « \n » pour un retour à la ligne et « \t » pour une tabulation. Nous les utiliserons dans des exemples de code plus tard.

Assignation contre égalité

Nous devons apprendre encore quelques petites choses avant d'être capable de faire notre prochain exemple. D'abord la différence entre l'assignation et l'égalité. Nous avons utilisé l'assignation de nombreuses fois dans nos exemples. Lorsqu'on veut assigner l'opérateur d'assignation ou « = » (le signe égal) :

```
variable = valeur
```

Cependant lorsqu'on veut estimer une variable et sa valeur, on doit utiliser un opérateur de comparaison. Supposons que nous voulions vérifier qu'une variable est égale à une valeur donnée. Nous devons utiliser le symbole « == » (deux signes égal) :

```
variable == valeur
```

Donc si nous avons une variable nommée « boucle » et que nous voulons savoir si elle est égale à, disons 12, nous utiliserions :

```
if boucle == 12:
```

Ne vous préoccupez pas encore du « if » et des deux-points dans l'exemple ci-dessus. Souvenez-vous simplement qu'on doit utiliser le signe « double-égal » pour faire une évaluation.

Commentaires

Nous allons maintenant discuter des commentaires. Les commentaires sont importants pour plusieurs raisons. Ils donnent non seulement une indication à vous ou à d'autres de ce que vous essayez de faire, mais quand vous reprenez votre code, disons 6 mois plus tard, ils peuvent vous aider à vous souvenir de ce que vous essayiez de faire. Quand vous commencez à écrire de nombreux programmes, cela devient important. Les commentaires vous permettent également d'indiquer à Python de ne pas tenir compte d'un certain nombre de lignes de code. Pour commenter une ligne, utilisez le signe « # ». Par exemple :

```
# Ceci est un commentaire
```

Vous pouvez commenter n'importe quelle ligne de code, souvenez-vous que Python ignorera tout ce qui ce trouve après « # »

Instructions « if »

Maintenant, retournons à l'instruction « if » que nous avons aperçue ci-dessus. Quand nous voulons prendre une décision basée sur la valeur d'une chose, nous pouvons utiliser l'instruction « if » :

```
 if boucle == 12:
```

La variable « boucle » sera vérifiée et si sa valeur est 12, tout ce qui se trouve dans le bloc indenté en dessous sera exécuté. La plupart du temps cela sera suffisant mais comment faire si nous voulons dire : si une variable vaut ça, alors faire ceci sinon faire à cela. En pseudo code, cela donnerait :

```
if x == y alors
      faire ceci
sinon
faire cela
```

et en Python, nous écririons :

```
if x == y:
          faire ceci
else:
          faire cela
          et d'autres choses
```

Les principales choses dont il faut se souvenir ici sont :

1. de terminer les instructions if et else par deux points,

2. d'INDENTER vos lignes de code.

Supposons que vous avez plus d'une chose à contrôler, vous pouvez utiliser le format if/elif/else. Par exemple :

```
x = 5
if x == 1:
print 'X vaut 1'
elif x < 6:
print 'X est plus petit
que 6'
elif x < 10:
print 'X est plus petit
10'
else:
print 'X est supérieur ou
égal à 10'
```

Notez que nous utilisons l'opérateur « < » pour voir si x est PLUS PETIT QU'une valeur, dans ce cas 6 et 10. D'autres opérateurs classiques de comparaisons sont « plus grand que

(>) », « inférieur ou égal à (<=) », « supérieur ou égal à (>=) » et « différent de (!=) ».

Instructions « while »

Finalement, regardons un exemple simple d'instruction « while ». L'instruction « while » vous permet de créer une boucle réalisant une série d'étapes encore et encore jusqu'à ce qu'un seuil spécifique soit atteint. Un exemple simple consiste à assigner une variable « compteur » à 1. Puis tant que la variable « compteur » est inférieure ou égale à 10, afficher la valeur de « compteur », ajouter un à celle-ci et continuer... jusqu'au moment où « compteur » est plus grande que 10, quitter :

```
compteur = 1
while compteur <= 10:
      print compteur
      compteur = compteur + 1
```

exécuté dans un terminal cela produirait la sortie suivante :

```
1
2
3
4
5
6
7
8
9
10
```

C'est exactement ce que nous voulions voir. La figure 1 (en haut à droite) est un exemple similaire, un peu plus compliqué, mais toujours simple.

Dans cet exemple, nous combinons l'instruction « if », la boucle « while », l'instruction « raw_input », la séquence d'échappement « retour à la ligne », l'opérateur d'assignation et l'opérateur de comparaison, tout cela en 8 lignes de programme.

L'exécution de ce programme produirait :

```
Saisissez quelque chose ou
'fin' pour sortir => LION
Vous avez saisi LION
Saisissez quelque chose ou
'fin' pour sortir => rat
Vous avez saisi rat
Saisissez quelque chose ou
'fin' pour sortir => 42
Vous avez saisi 42
Saisissez quelque chose ou
'fin' pour sortir => FIN
Vous avez saisi FIN
Saisissez quelque chose ou
'fin' pour sortir => fin
C'est fini
```

Notez que lorsque vous avez saisi « FIN », le programme ne s'est pas arrêté. C'est parce que nous comparons la valeur de la variable « reponse » à « fin » (reponse == 'fin'). « FIN » n'est pas égal à « fin ».

```
Boucle = 1
while boucle == 1:
      reponse = raw_input("Saisissez quelque chose ou 'fin' pour sortir => ")
      if reponse == 'fin':
            print "C'est fini"
            boucle = 0
      else:
            print 'Vous avez saisi %s' % reponse
```

Programme 1

Avant de se séparer ce mois-ci voici un dernier exemple rapide. Supposons que vous vouliez vérifier qu'un utilisateur est bien autorisé à accéder à votre programme. Bien que cet exemple ne soit pas la meilleure façon d'accomplir cette tâche, c'est une bonne manière d'illustrer ce que nous avons déjà appris. En gros, nous allons demander à l'utilisateur de saisir son nom et son mot de passe, les comparer avec les informations que nous avons codées dans le programme et prendre une décision basée sur nos résultats. Nous allons utiliser deux listes, l'une contient les utilisateurs autorisés et l'autre les mots de passe. Ensuite, nous utilisons raw_input pour récupérer les informations concernant l'utilisateur et finalement les instructions if/elif/else pour vérifier et décider de l'autorisation de l'utilisateur. Souvenez-vous, ce n'est pas la meilleure façon de procéder. Nous verrons d'autres possibilités dans les articles à venir. Notre code se trouve sur la page suivante, à droite.

Enregistrez-le sous le nom test_mot2passe.py » et lancez-le en essayant diverses possibilités.

La seule chose dont nous n'avons pas encore parlé, c'est la procédure de vérification de la liste commençant par « if usr in utilisateurs: ». Nous vérifions ainsi que le nom de l'utilisateur qui a été saisi, est présent dans la liste.

Dans ce cas, nous recherchons la position de son nom dans la liste « utilisateurs ». Nous utilisons pour cela l'instruction « utilisateurs.index(usr) » pour récupérer cette position afin d'extraire le mot de passe, enregistré à la même position dans la liste « mot2passe ». Par exemple, John est à la position 1 dans la liste « utilisateurs ». Son mot de

passe, « chien » est en position 1 dans la liste « mot2passe ». De cette façon, nous pouvons faire correspondre les deux. Cela devrait être facile à comprendre dans l'état actuel de vos connaissances.

C'est assez pour ce mois-ci. La prochaine fois, nous apprendrons les fonctions et les modules. En attendant, jouez avec ce que vous avez déjà appris et amusez-vous bien.

```
# coding=utf-8 (NDT : codage à adapter en fonction\\
# de la configuration de votre éditeur)
#-----------------------------------------------
#test_mot2passe.py
#     exemple de if/else, listes, assignations, raw_input,
#     commentaires et évaluations
#-----------------------------------------------
# Assigne les utilisateurs et les mots de passe
utilisateurs = ['Fred','John','Steve','Anne','Marie']
mot2passe = ['accès','chien','12345','enfants','azerty']
#-----------------------------------------------
# Récupérer les noms d'utilisateurs et les mots de passe
usr = raw_input("Saisissez votre nom d'utilisateur => ")
m2p = raw_input('Saisissez votre mot de passe => ')
#-----------------------------------------------
# Vérifier que l'utilisateur est dans la liste
if usr in utilisateurs:
      position = utilisateurs.index(usr) #Récupère la position de l'utilisateur
      if m2p == mot2passe[position]: #Cherche le mot de passe numéro position
            print 'Salut %s. Accès autorisé.' % usr
      else:
            print 'Mot de passe incorrect. Accès refusé.'
      else:
            print "Désolé... Je ne vous reconnais pas. Accès refusé."
```

Programme 2

TUTORIEL

Greg Walters

Programmer en Python - Partie 3

Dans l'article précédent, nous avons vu les listes, la substitution littérale, les commentaires, l'égalité et l'assignation, les instructions if et while. Je vous avais promis que dans cette partie nous aborderions les modules et les fonctions. Alors, allons-y.

Les modules

Les modules sont un moyen d'étendre votre programmation Python. Vous pouvez créer vos propres modules, utiliser ceux qui sont fournis avec Python, ou encore utiliser ceux que d'autres ont créés. Python est livré avec des centaines de modules divers qui rendent la programmation plus facile. Une liste des modules globaux fournis avec Python peut être consultée ici : http://docs.python.org/modindex.html.

Certains modules sont spécifiques au système d'exploitation mais la plupart sont complètement portables d'une plateforme à une autre (on peut les utiliser de la même façon sous Linux, Mac et Microsoft Windows). Pour utiliser un module externe, vous devez l'importer dans votre programme. L'un des modules livrés avec Python s'appelle « random ». Ce module vous permet de générer des nombres pseudo-aléatoires. Nous utiliserons le module (à droite) dans notre premier exemple.

```
#=====================================
# random_exemple.py
# Exemple d'utilisation d'un module : le module random
#=====================================
import random
# affiche 14 nombres aléatoires
for cntr in range(1,15):
print random.randint(1,10)
```

Examinons chaque ligne de code. Les quatre premières lignes sont des commentaires. Nous en avons parlé dans l'article précédent. La ligne 5 dit à Python d'utiliser le module « random ». Nous devons le préciser explicitement.

La ligne 7 démarre une boucle « for » pour afficher 14 nombres aléatoires. La ligne 8 utilise la fonction randint() pour afficher un entier aléatoire compris entre 1 et 10. Notez que l'on doit indiquer à Python le module dont provient la fonction. On fait ça en écrivant (dans ce cas) random.randint. Pourquoi créer des modules ? Eh bien, si toutes les fonctions étaient directement intégrées dans Python, non seulement Python deviendrait vraiment énorme et lent mais la correction des bogues deviendrait un vrai cauchemar. En utilisant des modules, on peut découper le code en morceaux qui répondent à un besoin spécifique. Si, par exemple, vous n'utilisez pas les fonctionnalités liées aux bases de données, vous n'avez pas besoin de savoir qu'il existe un module pour SQLite. Cependant, quand vous en aurez besoin, il sera déjà présent (d'ailleurs, nous utiliserons des modules de bases de données plus tard dans cette série d'articles).

Une fois bien habitué à programmer en Python, vous fabriquerez probablement vos propres modules pour pouvoir réutiliser maintes fois du code déjà écrit sans avoir à le retaper. Si vous devez changer quelque chose dans ce morceau de code, vous pourrez le faire avec très peu de risques de casser le code dans votre programme principal. Il y a des limites à cela sur lesquelles nous nous pencherons plus tard. Quand nous avons utilisé l'instruction « import random » précédemment, nous avons demandé à Python de nous donner accès à toutes les fonctions du module random. Si, au lieu de cela, nous avons seulement besoin de la fonction randint(), nous pouvons réécrire l'instruction d'importation ainsi :

```
from random import randint
```

Maintenant quand nous utilisons notre fonction, nous n'avons pas besoin d'utiliser l'identifiant « random. » devant. Donc, notre code devient :

```
from random import randint
# affiche 14 nombres
aléatoires
for cntr in range(1,15):
    print randint(1,10)
```

Les fonctions

Quand nous avons importé le module random, nous avons utilisé la fonction randint(). Une fonction est

un bloc de code conçu pour être appelé, en général plus d'une fois, ce qui le rend plus facile à maintenir, et nous évite de retaper sans cesse les mêmes extraits de code. Grosso modo, à chaque fois qu'on a à taper un bout de code plus d'une fois ou deux, ce bout de code est un bon candidat pour devenir une fonction. Bien que les deux exemples qui suivent soient simplistes, ils montrent bien comment utiliser une fonction. Disons que l'on veut prendre deux nombres, les ajouter, puis les multiplier, puis les soustraire, en affichant les résultats à chaque fois. Et pour compliquer les choses, nous devons faire ça trois fois avec des nombres différents. Voir notre exemple simpliste en haut à droite.

Non seulement cela fait beaucoup de lignes à taper, mais cela conduit également à des erreurs, soit lors de la saisie soit lors de changements ultérieurs. Au lieu de cela, nous allons créer une fonction appelée « Calcul Deux » qui prend les deux nombres et fait les calculs, affichant les résultats à chaque fois.

Nous commençons par utiliser le mot clé « def » (qui indique que l'on va définir une fonction). Après « def », nous ajoutons le nom choisi pour la fonction puis la liste des paramètres (s'il y en a) entre parenthèses. La ligne est terminée par deux points « : ». Le code de la fonction est indenté. Notre exemple simpliste amélioré (n°2) est visible ci-dessous.

```
# exemple simpliste
print 'Ajouter les deux nombres %d et %d = %d ' % (1,2,1+2)
print 'Multiplier les deux nombres %d et %d = %d ' % (1,2,1*2)
print 'Soustraire les deux nombres %d et %d = %d ' % (1,2,1-2)
print '\n'
print 'Ajouter les deux nombres %d and %d = %d ' % (1,4,1+4)
print 'Multiplier les deux nombres %d et %d = %d ' % (1,4,1*4)
print 'Soustraire les deux nombres %d et %d = %d ' % (1,4,1-4)
print '\n'
print 'Ajouter les deux nombres %d and %d = %d ' % (10,5,10+5)
print 'Multiplier les deux nombres %d et %d = %d ' % (10,5,10*5)
print 'Soustraire les deux nombres %d et %d = %d ' % (10,5,10-5)
print '\n'
```

Comme vous le voyez, il y a beaucoup moins de choses à taper — 8 lignes au lieu de 12. Si nous devons modifier la fonction, nous pouvons le faire sans risquer de causer trop de problèmes dans le programme principal. On appelle notre fonction, dans ce cas, en utilisant son nom suivi des paramètres.

Voici un autre exemple de fonction. Considérons les exigences suivantes.

Nous voulons créer un programme qui affiche une liste d'achats avec une jolie mise en forme. Cela devra ressembler au texte ci-après (bas de page suivante à gauche).

Le coût de chaque objet et le total général seront formatés en euros et centimes. La largeur du tableau doit pouvoir être modifiée. Les valeurs à gauche et à droite doivent être également variables. Nous utiliserons trois fonctions pour effectuer cette tâche. L'une affiche les lignes du haut et du bas, une autre affiche les détails des objets, y compris le total général, et la dernière affiche la ligne de séparation. Heureusement, Python rend tout cela très simple à réaliser. Souvenez-vous, nous avons affiché une chaîne multipliée par 4 et cela a retourné quatre copies de la même chaîne. Eh bien, nous pouvons réutiliser cela. Pour afficher la première et la dernière ligne on peut prendre la largeur désirée, retrancher 2 pour les 2 caractères + et utiliser « '=' * (largeur-2) ». Pour rendre les choses encore plus simples, nous utiliserons la substitution de variable pour mettre tous ces objets sur la même ligne. Ainsi notre chaîne d'affichage devient '%s%s%s' % ('+',('=' * largeur-2)),'+'). Maintenant on pourrait demander à la routine d'afficher ceci

```
# exemple simpliste 2... toujours simpliste mais un peu mieux
def CalculDeux(num1,num2):
     print 'Ajouter les deux nombres %d et %d = %d ' % (num1,num2,num1+num2)
     print 'Multiplier les deux nombres %d et %d = %d ' % (num1,num2,num1*num2)
     print 'Soustraire les deux nombres %d et %d = %d ' % (num1,num2,num1-num2)
     print '\n'
CalculDeux(1,2)
CalculDeux(1,4)
CalculDeux(10,5)
```

directement, mais nous utiliserons le mot clé return pour renvoyer la chaîne générée au programme appelant. Nous appellerons notre fonction « HautOuBas » et le code de cette fonction ressemble à ceci :

```
def HautOuBas(largeur):
      # largeur est la
largeur totale de la ligne
retournee
      return '%s%s%s' %
('+',('=' * (largeur-2)),'+')
```

Nous pourrions enlever le commentaire mais c'est pratique de pouvoir savoir d'un coup d'oeil ce que le paramètre « largeur » représente. Pour utiliser la fonction, nous utiliserions « print HautOuBas(40) » ou n'importe quelle largeur souhaitée. Maintenant, nous avons une fonction qui prend en charge la première et la dernière ligne. Nous pouvons créer une nouvelle fonction pour gérer la ligne séparatrice en utilisant le même genre de code... OU BIEN nous pouvons modifier la fonction que nous venons d'écrire en lui ajoutant un paramètre précisant le caractère à utiliser entre les caractères plus. Faisons cela. Nous pouvons garder le nom HautOuBas.

```
def
HautOuBas(caractere,largeur):
      # largeur est la
largeur totale de la ligne
retournee
      # caractere est le
caractere a placer entre les
'+'
      return '%s%s%s' %
('+',(caractere * (largeur-
2)),'+')
```

Maintenant vous pouvez voir l'utilité des commentaires. Rappelez-vous, on renvoie la chaîne générée donc on doit prévoir quelque chose pour la recevoir lorsqu'on appelle la fonction. Au lieu de l'assigner à une autre chaîne, nous allons directement l'afficher. Voici la ligne qui appelle notre fonction :

```
print HautOuBas('=',40)
```

Maintenant donc, non seulement on s'est occupé de 3 des lignes, mais on a également diminué le nombre de routines de 3 à 2. Il ne reste donc que la partie centrale de l'affichage à faire.

Appelons la nouvelle fonction « Fmt ». Nous lui passerons quatre paramètres :
val1 – la valeur à afficher à gauche,
largeurGauche – la largeur de cette « colonne »,
val2 – la valeur à afficher à droite (qui doit être un nombre contenant une virgule),
largeurDroite – la largeur de cette « colonne ».

La première chose à faire est de formater l'information de la partie droite. Puisque nous voulons afficher des euros et des centimes, nous pouvons utiliser une fonctionnalité spéciale de la substitution variable qui affiche la valeur comme un nombre réel avec n décimales. La commande serait '%2.f'. Nous allons assigner cette valeur à une variable « part2 ». Ainsi, notre ligne de code serait « part2 = '%2.f' % val2 ». Nous pouvons aussi utiliser les fonctions ljust et rjust fournies avec Python. Ljust justifie une chaîne à gauche, remplissant le côté droit avec un caractère de votre choix. Rjust fait la même chose mais le remplissage se fait à gauche. Maintenant la partie rusée. En utilisant les substitutions, nous créons une grande chaîne et la retournons au programme appelant. Voici notre prochaine ligne :

```
return '%s%s%s%s' % ('|
',val1.ljust(largeurGauche-
2,''),part2.rjust(largeurDro
i te-2,' '),' |')
```

Ceci paraît obscur de prime abord, alors expliquons les choses une par une et voyons comme c'est simple :
Return - nous renvoyons la chaîne créée au programme appelant,
'%s%s%s%s' - nous allons rassembler 4 valeurs dans notre chaîne. Chaque s est un substituant,
% (- début de la liste des variables,
'|', - affiche exactement cette chaîne,
val1.ljust(largeurGauche-2,' ') - prend la variable val1 passée en paramètre, la justifie à gauche avec des espaces sur (largeurGauche-2) caractères. Nous retranchons 2 à cause du '|' à gauche,
Part2.rjust(largeurDroite-2,' ') - justifie à droite la chaîne formatée indiquant le prix sur largeurDroite-2 caractères,
' |' - pour terminer la chaîne.

Et c'est tout ce qu'il y a à faire. Nous devrions vraiment faire du contrôle d'erreurs, mais je vous laisse le faire par vous même. Donc... notre fonction Fmt ne fait que deux lignes de code en dehors de la définition et des commentaires. Nous pouvons l'utiliser ainsi :

```
print
Fmt('Objet
1',30,objet1,10)
```

```
'+==============================+'
 '| Objet 1                X.XX|'
 '| Objet 2                X.XX|'
 '+------------------------------+'
 '| Total                  X.XX|'
'+==============================+'
```

Encore une fois, nous pourrions assigner la valeur retournée à une autre chaîne mais nous allons simplement l'afficher. Notez que nous envoyons 30 comme largeur pour la colonne de gauche et 10 pour la colonne de droite. Ce qui fait 40 de largeur totale à envoyer à la fonction HautOuBas. Alors ouvrez votre éditeur et saisissez le code situé ci-dessous à droite. Enregistrez le code dans le fichier « pprint1.py » et exécutez-le. La sortie devrait ressembler à l'image ci-dessous, à droite. Cet exemple est très simple, mais il devrait vous donner une bonne idée de comment et pourquoi utiliser des fonctions. Maintenant, élargissons un peu tout cela et apprenons-en un peu plus sur les listes. Vous vous rappelez dans la deuxième partie quand nous avons parlé des listes ? Une des choses que je ne vous ai pas dite est qu'une liste peut contenir n'importe quoi, y compris des listes. Définissons une nouvelle liste dans notre programme nommée objs et remplissons-la ainsi :

```
objs =
[['Soda',1.45],['Bonbons',.75
],['Pain',1.95],['Lait',2.59]
]
```

Si on l'utilisait comme une liste normale, on ferait print objs[0]. Cependant, on obtiendrait ['Soda',1.45], ce qui n'est pas vraiment ce qu'on recherche habituellement. Nous voulons avoir accès à chaque objet de la première liste. Donc il faudrait écrire « objs[0][0] » pour obtenir « Soda » et [0][1] pour obtenir le prix, soit 1.45. Nous avons donc maintenant quatre objets qui ont été achetés et nous voulons utiliser cette information dans notre belle routine. La seule chose à changer est la fin du programme. Enregistrez le programme précédent sous le nom « pprint2.py » puis commentez les deux définitions d'objets et insérez la liste au-dessus. Cela devrait vous donner :

```
#objet1 = 3.00
#objet2 = 15.00
objs =
[['Soda',1.45],['Bonbons',.75],
['Pain',1.95],['Lait',2.59]]
```

```
+======================================+
| Objet 1                        3.00 |
| Objet 2                       15.00 |
+--------------------------------------+
| Total                         18.00 |
+======================================+
```

Puis supprimez tous les appels à Fmt(). Ensuite, ajoutez les lignes suivantes (celles se terminant par #NOUVEAU) pour que votre code ressemble à celui en haut de la page suivante. J'utilise un compteur pour faire une boucle sur les objets de la liste. Notez que j'ai aussi ajouté une variable appelée total. On initialise le total à 0 avant de rentrer dans la boucle. Puis au fur et à mesure qu'on affiche chaque objet vendu, on ajoute son prix à notre total. Finalement, on affiche le total juste après la ligne séparatrice. Sauvez votre programme et exécutez-le. Vous devriez voir

```
#pprint1.py
#Exemple de fonctions un peu utiles

def HautOuBas(caractere,largeur):
      # largeur est la largeur totale de la ligne retournee
         return '%s%s%s' % ('+',(caractere * (largeur-2)),'+')

def Fmt(val1,largeurGauche,val2,largeurDroite):
      # affiche deux valeurs justifiees par des espaces
      # val1 est a afficher a gauche, val2 est a afficher a droite
      # largeurGauche est la largeur de la partie gauche, largeurDroite est la largeur de la partie droite
       part2 = '%.2f' % val2
       return '%s%s%s%s' % ('| ',val1.ljust(largeurGauche-2,' '),part2.rjust(largeurDroite-2,' '),' |')
# definit le prix de chaque objet
objet1 = 3.00
objet2 = 15.00
# affiche tout
print HautOuBas('=',40)
print Fmt('Objet 1',30,objet1,10)
print Fmt('Objet 2',30,objet2,10)
print HautOuBas('-',40)
print Fmt('Total',30,objet1+objet2,10)
print HautOuBas('=',40)
```

quelque chose comme l'image du tableau ci-dessous, à droite, après le code. Si vous vous sentez l'âme d'un aventurier, vous pouvez ajouter une ligne pour la TVA. Inspirez-vous de ce qu'on a fait pour la ligne du total, mais utilisez (total * 0.196) pour le calcul de la TVA.print Fmt('TVA:',30,total*.196,10) Si vous voulez, vous pouvez ajouter d'autres objets dans la liste et voir comment cela fonctionne. C'est tout pour cette fois-ci. Au prochain numéro nous nous concentrerons sur les classes.

Amusez-vous bien.

```
objs =
[['Soda',1.45],['Bonbons',.75],['Pain',1.95],['Lait',2.59]]

print HautOuBas('=',40)

total = 0 #NOUVEAU
for ctr in range(0,4): #NOUVEAU
     print Fmt(objs[ctr][0],30,objs[ctr][1],10) #NOUVEAU
     total += objs[ctr][1] #NOUVEAU
print HautOuBas('-',40)
print Fmt('Total',30,total,10) #MODIFIEE
print HautOuBas('=',40)
```

```
+======================================+
| Soda                            1.45 |
| Bonbons                         0.75 |
| Pain                            1.95 |
| Lait                            2.59 |
+--------------------------------------+
| Total                           6.74 |
+======================================+
```

TUTORIEL

Greg Walters

Programmer en Python - Partie 4

La dernière fois je vous ai promis qu'on parlerait des classes. Alors, c'est là-dessus que nous allons nous concentrer aujourd'hui. Que sont les classes et qu'apportent-elles ?

Une classe permet de construire des objets. Un objet est simplement un moyen de manipuler des attributs et des comportements de façon globale. Je sais que cela peut paraître confus, mais je vais vous l'expliquer en détail. Voyez-le comme ceci : un objet est un moyen de modéliser quelque chose qui appartient au monde réel. Une classe est un moyen d'implémenter cette modélisation. Par exemple, nous avons trois chiens à la maison, un beagle, un labrador, et un croisement de berger allemand et de bouvier australien. Tous les trois sont des chiens, mais ils sont tous différents. Il y a des attributs communs aux trois, mais chaque chien a aussi des attributs propres. Par exemple, le beagle est petit, potelé, marron et grognon. Le labrador est de taille moyenne, noir et très décontracté. Le croisé berger allemand/bouvier est grand, maigrichon, noir et un peu dingue. Certains attributs sautent immédiatement aux yeux. Petit/de taille moyenne/grand sont des attributs de taille. Grognon/décontracté/dingue sont des attributs d'humeur. Quant au comportement, on peut examiner leur façon de manger, de dormir, de jouer et autres.

```
class Chien():
      def
__init__(self,nomChien,couleurChien,tailleChien,corpulenceChien,humeurChien,ageChien):
          # ici nous paramétrons les attributs de notre chien
          self.nom = nomChien
          self.couleur = couleurChien
          self.taille = tailleChien
          self.corpulence = corpulenceChien
          self.humeur = humeurChien
          self.age = ageChien
          self.AFaim = False
          self.ASommeil = False
```

Tous les trois appartiennent à la classe « Chien ». Pour revenir aux attributs utilisés pour décrire chacun, nous avons des valeurs telles que Chien.Nom, Chien.Taille, Chien.Corpulence (maigrichon, potelé, etc.), et Chien.Couleur. Nous avons aussi des comportements tels que Chien.Aboyer, Chien.Manger, Chien.Dormir et ainsi de suite.

Comme je l'ai déjà dit, chaque chien est d'une race différente. Chaque race serait une sous-classe de la classe Chien. Sur un diagramme, cela donnerait ceci :

```
          /--Beagle
Chien ---|-- Labrador
          \--Bouvier
```

Chaque sous-classe hérite de tous les attributs de la classe Chien. Ainsi, si on crée une instance de Beagle, elle obtient tous les attributs de sa classe mère, Chien.

```
Beagle = Chien()
Beagle.Nom = 'Archie'
Beagle.Taille = 'Petit'
Beagle.Corpulence = 'Potelé'
Beagle.Couleur = 'Marron'
```

Cela devient plus compréhensible ? Bon, créons notre classe Chien (voir ci-dessus). Nous commençons avec le mot-clé « class » et le nom de la classe.

Avant d'aller plus loin avec le code, notez la fonction que l'on a défini ici. La fonction __init__ (deux symboles souligné + « init » + deux symboles souligné) est une fonction d'initialisation qui fonctionne avec n'importe quelle classe. Dès que nous utilisons notre classe dans le code, cette routine est exécutée. Dans notre cas, nous avons prévu un certain nombre de paramètres correspondant aux informations de base concernant notre classe : nous avons un nom, une couleur, une taille, une corpulence, une humeur, un âge et

deux variables AFaim et ASommeil. Nous reviendrons là-dessus plus tard. Pour l'instant, ajoutons un peu plus de code :

```
Beagle = Chien('Archie','Marr
on','Petit','Potelé','Grognon
',12)
print Beagle.nom
print Beagle.couleur
print Beagle.humeur
print Beagle.AFaim
```

Ceci est du code NON INDENTÉ qui se trouve en dehors de notre classe, le code qui utilise notre classe. La première ligne crée une instance de notre classe chien appelée Beagle. On appelle cela l'instanciation. En la faisant, nous avons également passé quelques informations à l'instance de la classe, notamment le nom du chien, sa couleur, etc. Les quatre lignes suivantes interrogent simplement l'objet Beagle et reçoivent des informations en retour. Il est temps d'écrire encore un peu de code. Ajoutez le code situé en haut à droite dans la classe après la fonction __init__.

Maintenant nous pouvons appeler Beagle.Manger() ou Beagle.Dormir(). Ajoutons encore une méthode. Nous l'appellerons Aboyer. Son code se trouve à droite.

Celle-ci est plus flexible. Suivant l'humeur du chien, l'aboiement changera. Sur la page suivante, vous trouverez le code complet de la classe jusqu'ici. Quand on exécute tout cela, on obtient :

```
Mon nom est Archie
Ma couleur est Marron
Mon humeur est Grognon
J'ai faim = False
Sniff Sniff... Pas faim
Yum Yum...Num Num
GRRRRR...Woof Woof
```

Voilà pour mon bon vieux beagle grognon. Cependant, j'ai dit plus tôt que j'avais trois chiens. Puisque nous avons codé la classe avec soin, tout ce qu'il nous reste à faire est de créer deux nouvelles instances de notre classe chien.

```
Lab = Chien('Nina','Noir','Mo
yen','Lourd','Calme',7)
Bouvier = Chien('Bear','Noir'
,'Grand','Maigrichon','Dingue
',9)
print 'Mon nom est %s' %
Lab.nom
print 'Ma couleur est %s' %
Lab.couleur
print 'Mon humeur est %s' %
Lab.humeur
print "J'ai faim = %s" %
Lab.AFaim
Lab.Aboyer()
Bouvier.Aboyer()
```

```
def Manger(self):
     if self.AFaim:
          print 'Yum Yum...Num Num'
          self.AFaim = False
     else:
          print 'Sniff Sniff... Pas faim'
def Dormir(self):
     print 'ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'
     self.ASommeil = False
```

```
def Aboyer(self):
     if self.humeur == 'Grognon':
               print 'GRRRRR...Woof Woof'
     elif self.humeur == 'Calme':
          print 'Je baille...ok...Woof'
     elif self.humeur == 'Dingue':
          print "J'aboie, j'aboie, j'aboie, j'aboie,
j'aboie, j'aboie, j'aboie"
     else:
          print 'Woof Woof'
```

Notez que j'ai créé les instances des deux chiens avant de faire les affichages. Ceci n'est pas un problème, puisque j'ai « défini » les instances avant d'appeler l'une des méthodes. Voici la sortie complète de notre programme :

```
Mon nom est Archie
Ma couleur est Marron
Mon humeur est Grognon
J'ai faim = False
Sniff Sniff... Pas faim
Yum Yum...Num Num
GRRRRR...Woof Woof
Mon nom est Nina
Ma couleur est Noir
Mon humeur est Calme
J'ai faim = False
Je baille...ok...Woof
J'aboie, j'aboie, j'aboie,
j'aboie, j'aboie, j'aboie,
j'aboie
```

Maintenant, vous avez les bases, et votre travail personnel sera d'étendre la classe chien en ajoutant d'autres méthodes, comme peut-être Jouer ou RencontrerUnChienInconnu ou quelque chose de similaire.

La prochaine fois, nous commencerons à parler de la programmation

```
# coding=utf-8 (NDT : codage à adapter en fonction
# de la configuration de votre éditeur)
class Chien():
  def __init__(self,nomChien,couleurChien,tailleChien,comportementChien,hum
eurChien,ageChien):
      # ici nous paramétrons les attributs de notre chien
      self.nom = nomChien
      self.couleur = couleurChien
      self.taille = tailleChien
      self.comportement = comportementChien
      self.humeur = humeurChien
      self.age = ageChien
      self.AFaim = False
      self.ASommeil = False

  def Manger(self):
      if self.AFaim:
            print 'Yum Yum...Num Num'
            self.AFaim = False
      else:
            print 'Sniff Sniff... Pas faim'

  def Dormir(self):
      print 'ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'
      self.ASommeil = False

  def Aboyer(self):
      if self.humeur == 'Grognon':
            print 'GRRRRR...Woof Woof'
      elif self.humeur == 'Calme':
            print 'Je baille...ok...Woof'
      elif self.humeur == 'Dingue':
            print "J'aboie, j'aboie, j'aboie, j'aboie, j'aboie, j'aboie, j'aboie"
      else:
            print 'Woof Woof'

Beagle = Chien('Archie','Marron','Petit','Potelé','Grognon',12)
print 'Mon nom est %s' % Beagle.nom
print 'Ma couleur est %s' % Beagle.couleur
print 'Mon humeur est %s' % Beagle.humeur
print "J'ai faim = %s" % Beagle.AFaim
Beagle.Manger()
Beagle.AFaim = True
Beagle.Manger()
Beagle.Aboyer()
```

d'une interface graphique utilisateur (GUI : Graphical User Interface) en utilisant Boa Constructor.

TUTORIEL

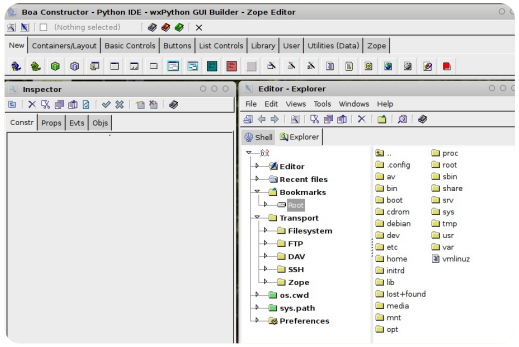
Programmer en Python - Partie 5

Greg Walters

Si vous êtes comme moi, vous allez détester la première partie de cette installation. Je déteste lorsqu'un auteur me dit de lire très attentivement chaque mot de leur livre / chapitre / article, parce que je sais tout de suite que cela va être très soporifique (même si je sais que c'est pour mon bien et qu'au final je le ferai de toute façon).

Je vous aurai prévenu. Veuillez lire avec attention les trucs ennuyeux qui suivent. Nous passerons ensuite aux choses amusantes, mais il faut d'abord poser les fondements avant de pouvoir parler réellement de programmation.

Premièrement, installez Boa Constructor et wxPython. Utilisez Synaptic et sélectionnez ces deux éléments. Une fois installés, vous devriez trouver Boa dans le menu Applications|Programmation|Boa Constructor. Allez-y, lancez l'application. Cela sera plus facile pour comprendre la suite. Une fois l'application démarrée, trois fenêtres (ou cadres, « frames » en anglais) différentes apparaissent : l'une remplissant le haut de l'écran et les deux autres en dessous. Vous devrez peut-être les redimensionner et les déplacer légèrement afin d'obtenir quelque chose qui ressemble à ceci :



La fenêtre du haut est appelée la palette des outils. Celle en bas à gauche est l'inspecteur et celle en bas à droite est l'éditeur. La palette présente différents onglets (Nouveau, Conteneurs/Mise en page, etc.) qui vous permettent de démarrer de nouveaux projets, d'ajouter des cadres à des projets existants et d'ajouter divers contrôles sur les fenêtres de votre application. L'inspecteur va devenir très important dès que nous ajouterons des contrôles à notre programme. L'éditeur nous permet de saisir notre code, d'enregistrer nos projets et plus encore. Intéressons-nous maintenant à la palette des outils et examinons chaque onglet, en commençant par l'onglet « Nouveau ». Bien qu'il y ait beaucoup d'options disponibles ici, nous n'en présenterons que deux. Ce sont les 5e et 6e boutons en partant de la gauche : « wx.App » et « wx.Frame ». Le bouton « wx.App » nous permet de créer une application complète en commençant par générer automatiquement deux fichiers. L'un est un fichier « cadre » (frame) et l'autre est un fichier application. C'est ma façon de procéder préférée. Le « wx.Frame » est utilisé pour ajouter d'autres cadres à notre application et/ou pour créer une application autonome à partir d'un seul fichier source. Nous en reparlerons plus tard.

Maintenant, observons l'onglet Conteneurs/Mise en page. Beaucoup de choix ici. Le plus utilisé sera le « wx.Panel » (le premier à gauche) et les « sizers » (2, 3, 4, 5 et 6 à partir de la droite). Sous l'onglet Composants de base, vous trouverez entre autres les contrôles de texte statique (les étiquettes), les boîtes à texte, les cases à cocher, les boutons radio. Sous l'onglet Boutons, vous trouverez diverses formes de boutons. Les contrôles Listes contiennent les tableaux de données et d'autres boîtes à liste. Passons maintenant à l'onglet Divers où vous trouverez les minuteurs (timers) et les éléments de menu.

Voici quelques petites choses dont il faut se souvenir avant d'attaquer notre première application. Il y a quelques bogues dans la version Linux. L'un d'eux est qu'il n'est pas possible de déplacer certains contrôles dans le concepteur. Utilisez les raccourcis <Ctrl> + flèches pour déplacer ou modifier légèrement la position de vos contrôles. Un autre, que vous découvrirez si vous essayez les tutoriels qui sont fournis avec Boa Constructor, est qu'il est difficile de positionner visuellement un contrôle de panneau. Recherchez les petites boîtes (je vous montrerai cela bientôt). Vous pouvez aussi utiliser l'onglet Objs de l'inspecteur et sélectionner l'objet de cette façon.

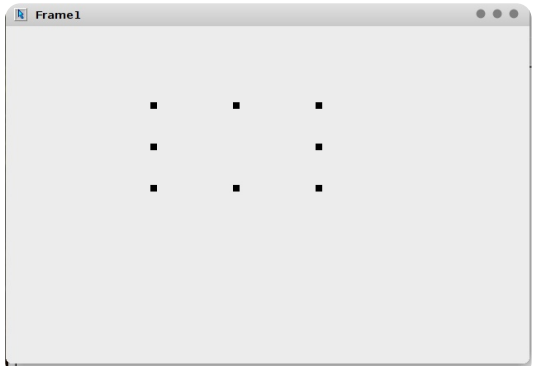
Ok, allons-y. Dans l'onglet Nouveau de la palette des outils, sélectionnez « wx.App » (5e bouton en partant de

la gauche). Cela va créer deux nouveaux onglets dans l'éditeur : l'un s'appelle « *(App1)* », l'autre « *(Frame1)* ». Incroyable, mais vrai, la **toute** première chose que nous allons faire est d'enregistrer les deux nouveaux fichiers, en commençant par le fichier Frame1. Le bouton pour enregistrer est le 5e bouton en partant de la gauche dans l'éditeur. Une fenêtre « Enregistrer sous » surgit pour vous demander où enregistrer le fichier et comment l'appeler. Créez un dossier dans votre répertoire personnel appelé TestsGui et enregistrez le fichier en tant que « Cadre1.py ». Notez que l'onglet « *(Frame1)* » s'appelle maintenant « Cadre1 » (les « *(» indiquent que le fichier a besoin d'être enregistré). Faites la même chose avec l'onglet App1.

Maintenant, examinons certains boutons de la barre d'outils de l'éditeur. Pour le moment, seuls nous importent Enregistrer (5e bouton en partant de la gauche) et « Démarrer l'application » (une flèche jaune, à la 7e place en partant de la gauche). Si vous êtes dans un onglet cadre (Cadre1, par exemple), vous devez apprendre à utiliser d'autres boutons. Commençons par le bouton « Éditeur graphique » :

Il est important car il nous permet de concevoir notre fenêtre de l'application graphique et c'est ce que nous allons faire maintenant. Si vous cliquez dessus, une fenêtre vide apparaît.

C'est un canevas vide sur lequel vous pouvez déposer tous les contrôles dont vous avez besoin (dans les limites du raisonnable). La première chose que nous voulons faire est d'y positionner un contrôle « wx.panel ». Dans presque tous les documents que j'ai lus, il est dit de ne pas mettre de contrôles (hormis un wx.panel) directement dans une fenêtre. Par conséquent, cliquez sur l'onglet Conteneurs/Mise en page de la palette puis cliquez sur le bouton « wx.Panel ». Ensuite, déplacez-vous vers la nouvelle fenêtre sur laquelle vous êtes en train de travailler et cliquez quelque part à l'intérieur de celle-ci. Vous saurez si cela a fonctionné si vous voyez quelque chose comme ceci :

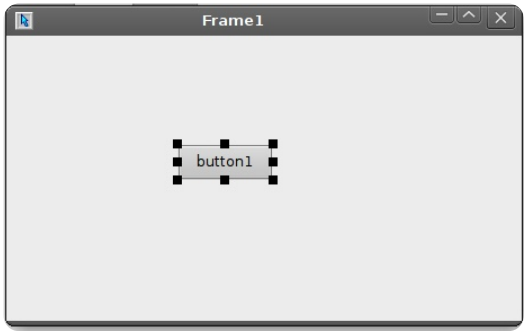


Vous vous rappelez quand je vous ai parlé des bogues ? Eh bien, en voici un. Ne vous inquiétez pas. Vous apercevez les 8 petits carrés noirs ? Ils représentent les bords du panneau. Si vous voulez, vous pouvez cliquer et faire glisser l'un d'eux pour redimensionner le panneau. Ce qui nous intéresse cette fois-ci, c'est d'avoir un panneau qui recouvre entièrement la fenêtre. Pour l'instant, redimensionnez seulement un peu la fenêtre. Maintenant nous disposons d'un panneau pour y placer nos autres contrôles. Déplacez la fenêtre sur laquelle vous travaillez afin de voir la barre d'outils de l'éditeur. Deux nouveaux boutons sont apparus : un symbole « Valider » et un « X ». Le « X » sert à annuler les modifications que vous avez faites.

Le bouton Valider : 

est appelé le bouton « Envoyer ». Il sert à écrire vos modifications dans le fichier cadre. Vous devrez quand même enregistrer le fichier cadre mais cela permet d'intégrer les nouvelles choses dans le fichier. Cliquez sur le bouton « Envoyer ». Un autre bouton « Envoyer » existe également dans la fenêtre de l'inspecteur, mais nous en reparlerons plus tard. Maintenant enregistrez votre fichier.

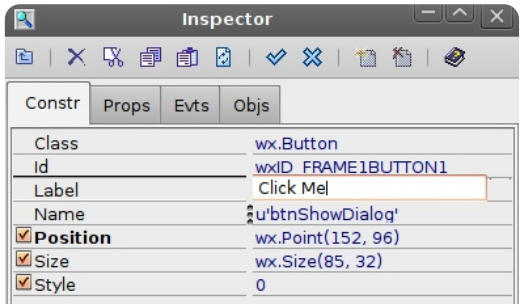
Retournez dans le mode Conception. Cliquez sur l'onglet « Boutons » dans la palette des outils, puis cliquez sur le premier bouton de gauche, le bouton « wx.Button ». Positionnez-le quelque part près du centre de votre cadre. Vous devez obtenir quelque chose comme ceci :



Remarquez qu'il est entouré de 8 petits carrés, tout comme le panneau. Ce sont des poignées de dimensionnement. Elles permettent également de savoir quel est le contrôle actuellement sélectionné. Pour placer le bouton plus près du centre du cadre,

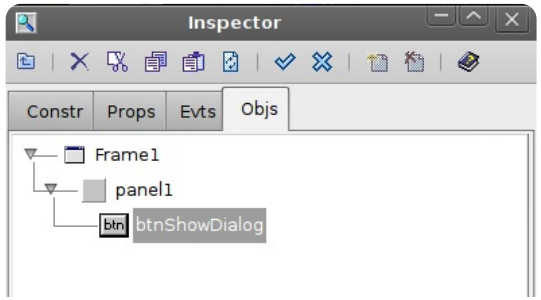
appuyez sur la touche Contrôle (Ctrl) et tout en la gardant enfoncée, utilisez les flèches du clavier pour le déplacer à votre guise.

Maintenant regardons l'inspecteur. Il y a quatre onglets. Cliquez sur l'onglet « Constr ». Dans celui-ci, vous pouvez modifier l'étiquette (Label), le nom (Name), la position, la taille (Size) et le style. Pour l'instant, changeons



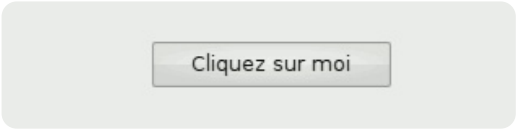
le nom en « btnAfficheDialog » et l'étiquette en « Cliquez sur moi ».

Pour l'instant, laissons tel quel le reste de cet onglet et allons dans l'onglet « Objs ». Cet onglet affiche tous les contrôles existants et leur



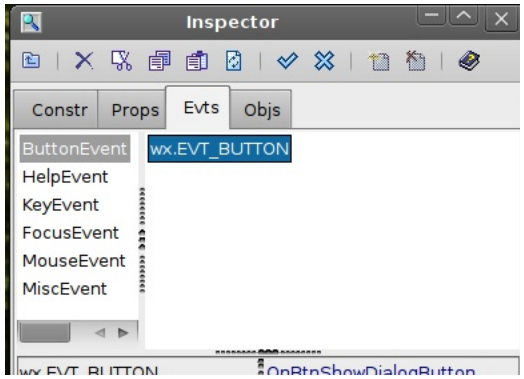
relation parent/enfant. Comme vous pouvez le voir, le bouton est un enfant du « panel1 », qui est un enfant du « Frame1 ».

Envoyez (bouton valider) et enregistrez vos modifications. Retournez au concepteur une fois de plus et remarquez que (en supposant que l'onglet « Objs » de l'inspecteur est toujours sélectionné) « Frame1 » est actuellement sélectionné. Tant mieux, puisque c'est ce que nous voulons. Retournez dans l'onglet « Constr » et



modifiez le titre (Title) « Frame1 » en « Notre premier GUI ». Envoyez et enregistrez une fois de plus. Maintenant, lançons notre application en cliquant sur le bouton jaune « Démarrer l'application » dans la fenêtre de l'éditeur.

Cliquez autant que vous pouvez sur le bouton, mais rien ne se produit. Pourquoi ? Eh bien, parce que nous n'avons pas dit au bouton de faire quoi que ce soit. Pour cela, nous devons paramétrer un événement qui doit se produire, ou se déclencher, lorsque l'utilisateur clique sur notre bouton. Cliquez sur le X dans le coin supérieur droit pour arrêter l'exécution de la fenêtre. Ensuite, retournez dans le concepteur, sélectionnez le bouton



et allez dans l'onglet « Evts » de l'inspecteur. Cliquez sur « ButtonEvent » puis double-cliquez sur le texte wx.EVT_BUTTON qui s'affiche et remarquez que nous obtenons un bouton appelé : « OnbtnAfficheDialogButton » dans la fenêtre en dessous. Envoyez et enregistrez.

Avant d'aller plus loin, examinons le code que nous obtenons (page suivante).

La troisième ligne est un commentaire qui indique à Boa Constructor que c'est un fichier boa. Il est ignoré par le compilateur Python, mais pas par Boa. La ligne suivante importe wxPython. Maintenant, sautons à la définition de la classe.

Au début, il y a la méthode « init_ctrls ». Notez le commentaire juste en dessous de la ligne de définition. Ne modifiez pas le code de cette fonction. Si vous le faites, vous vous en mordrez les doigts. Tout ce qui est en dessous de cette méthode doit rester intact. On y trouve les définitions de chaque contrôle de notre fenêtre.

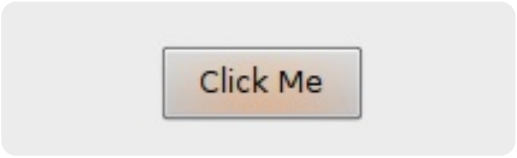
Ensuite, regardez la fonction «___init___». Vous pouvez placer ici n'importe quel appel à du code d'initialisation. Enfin, le bloc OnBtnAfficheDialogButton. C'est là que nous plaçons le code qui fera tout le travail lorsque l'utilisateur cliquera sur le bouton. Remarquez qu'il y a une ligne event.Skip() ici actuellement. Pour faire simple, elle indique juste que l'événement doit être ignoré lorsqu'il se déclenche.

Maintenant, ce que nous allons faire est un appel qui fera surgir une boîte de message avec un texte. C'est une chose que les programmeurs font couramment pour permettre à l'utilisateur d'avoir une information sur quelque chose : une erreur ou le fait qu'un processus est terminé. Dans notre cas, nous appellerons la routine intégrée « wx.MessageBox ». Le programme est appelé avec deux arguments. Le premier est le texte que nous souhaitons

envoyer dans la boîte de message et le second est son titre. Mettez en commentaire la ligne « event.Skip() » et saisissez la ligne suivante :

```
wx.MessageBox('Vous avez
cliqué sur le bouton',
'Info')
```

Enregistrez et cliquez sur le bouton « Démarrer l'application » (flèche jaune). Vous devriez voir quelque chose comme ceci :



Et quand vous cliquez sur le bouton, vous devriez voir quelque chose comme ceci :

Comprenez bien que ceci est la manière la plus simple d'appeler la routine : « wx.MessageBox ». Il peut y avoir plus de paramètres également.

Voici un bref sur-vol sur la façon de modifier le comportement des icônes sur la boîte de message (la suite la prochaine fois).

wx.ICON_QUESTION - Afficher une icône interrogation.

wx.ICON_EXCLAMATION - Afficher une icône alerte.

wx.ICON_ERROR - Afficher une icône erreur.

wx.ICON_INFORMATION - Afficher une icône information.

La façon d'écrire cela pourrait être :

```
wx.MessageBox('Vou
s avez cliqué sur
le bouton',
'Info',
wx.ICON_INFORMATION)
```

ou n'importe quelle icône qui correspond à la situation. Il existe aussi diverses méthodes d'arrangement de boutons dont nous parlerons la prochaine fois.

```
# coding=utf-8 (NDT : codage à adapter en fonction
# de la configuration de votre éditeur)
#Boa:Frame:Frame1
import wx
def create(parent):
    return Frame1(parent)
[wxID_FRAME1, wxID_FRAME1BNTAFFICHEDIALOG, wxID_FRAME1PANEL1,
] = [wx.NewId() for _init_ctrls in range(3)]

class Frame1(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit (méthode générée, ne pas modifier)
        wx.Frame.__init__(self, id=wxID_FRAME1, name='', parent=prnt,
              pos=wx.Point(558, 440), size=wx.Size(556, 427),
              style=wx.DEFAULT_FRAME_STYLE, title=u'Notre premier GUI')
        self.SetClientSize(wx.Size(556, 427))
        self.panel1 = wx.Panel(id=wxID_FRAME1PANEL1, name='panel1', parent=self,
              pos=wx.Point(0, 0), size=wx.Size(556, 427),
              style=wx.TAB_TRAVERSAL)
        self.bntAfficheDialog = wx.Button(id=wxID_FRAME1BNTAFFICHEDIALOG,
              label=u'Cliquez sur moi', name=u'bntAfficheDialog',
              parent=self.panel1, pos=wx.Point(136, 120), size=wx.Size(85, 29),
              style=0)
        self.bntAfficheDialog.Bind(wx.EVT_BUTTON, self.OnBntAfficheDialogButton,
              id=wxID_FRAME1BNTAFFICHEDIALOG)

    def __init__(self, parent):
        self._init_ctrls(parent)
    def OnBntAfficheDialogButton(self, event):
        event.Skip()
```

En attendant la suite, jouez donc avec quelques-uns des divers contrôles, positionnements, etc. Amusez-vous !

TUTORIEL

Programmer en Python - Partie 6

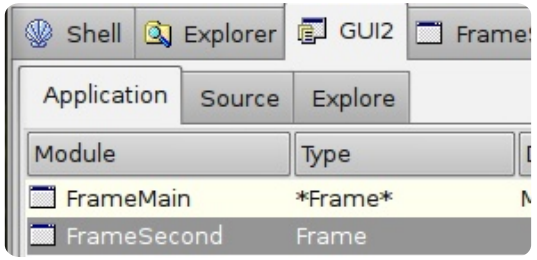
Greg Walters

J'espère que vous vous êtes bien amusés avec Boa Constructor depuis notre dernière rencontre. Nous commencerons par un programme très simple qui affiche une fenêtre et qui vous permet de cliquer sur un bouton afin de faire apparaître une autre fenêtre. La dernière fois, c'était une boîte de message. Cette fois-ci, nous allons créer une fenêtre totalement indépendante. Cela peut être très utile lors de la création d'une application qui contient plusieurs fenêtres (ou cadres). Bon, allons-y…

Démarrez Boa Constructor et fermez tous les onglets de l'éditeur, sauf les onglets shell et explorateur, en utilisant le raccourci clavier (Ctrl+W). De cette façon, vous êtes sûr de reprendre complètement à zéro. Maintenant, créez un nouveau projet en cliquant sur le bouton wx.App (consultez l'article précédent si nécessaire).

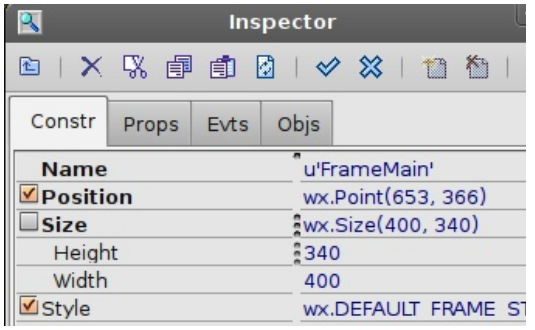
Avant de faire quoi que ce soit, enregistrez Frame1 sous le nom « FenetrePrincipale.py » puis enregistrez App1 sous le nom « GUI2.py ». C'est important. Après avoir sélectionné l'onglet GUI2 dans l'éditeur, allez dans la palette d'outils et ajoutez une autre fenêtre à votre projet en cliquant sur wx.Frame (qui se trouve juste à droite du bouton wx.App). Assurez-vous que l'onglet application de l'onglet GUI2 affiche les deux fenêtres dans la colonne module. Retournez dans la nouvelle fenêtre et enregistrez-la sous le nom « DeuxiemeFenetre.py ».

Ensuite, ouvrez FenetrePrincipale avec l'éditeur graphique et ajoutez un wx.panel à cette fenêtre. Redimensionnez-le un peu pour qu'il recouvre entièrement la fenêtre. Ensuite nous allons modifier quelques propriétés, ce que nous n'avions pas fait la dernière fois. Dans l'inspecteur, sélectionnez Frame1 dans l'onglet objs puis, dans l'onglet constr, paramétrez le titre (Title) à « Fenêtre Principale » et le nom (Name) à « FenetrePrincipale ». Nous parlerons des conventions pour l'utilisation des noms un peu plus tard. Réglez la taille à 400×340 en cliquant sur la case à cocher taille (Size). Cela fait apparaître la hauteur (Height) et la largeur (Width) en dessous. La hauteur devrait être 400 et la largeur 340.



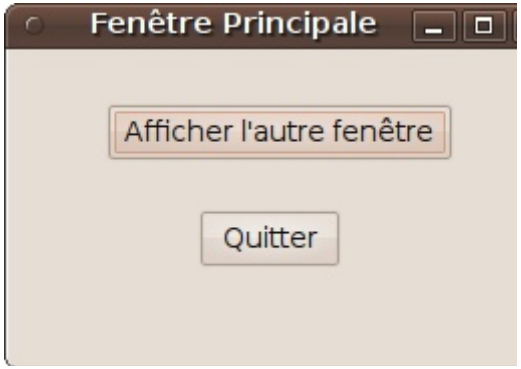
Maintenant cliquez sur l'onglet props. Cliquez sur la propriété « Centered » et paramétrez-la sur wx.BOTH. Cliquez sur l'icône Envoyez et enregistrez votre travail. Maintenant, lancez l'application en cliquant sur l'icône en forme de flèche jaune. Notre programme affiche une fenêtre au centre de l'écran dont le titre est « Fenêtre principale ». Ensuite, quittez en cliquant sur le « X » dans le coin supérieur droit de l'appli.

Ouvrons de nouveau FenetrePrincipale avec l'éditeur graphique. Ajoutez deux wx.Button dans la fenêtre, l'un



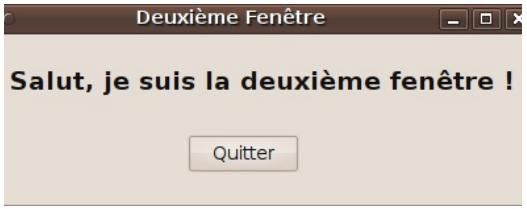
au-dessus de l'autre et près du centre de la fenêtre. Sélectionnez le bouton du haut et nommez-le « btnAfficheNouveau » puis saisissez son étiquette (Label) « Afficher l'autre fenêtre » dans l'onglet constr de l'inspecteur. Utilisez la combinaison Maj+Flèche pour redimensionner le bouton afin que tout le texte soit visible, puis la combinaison Ctrl+Flèche pour le repositionner au centre de la fenêtre. Sélectionnez le bouton du bas, nommez-le « btnQuitter » et saisissez « Quitter » pour l'étiquette. Envoyez, enregistrez et relancez pour voir les modifications. Quittez l'application et retournez dans l'éditeur graphique. Nous allons ajouter les événements de clic sur les boutons. Sélectionnez le bouton du haut et l'onglet Evts dans l'inspecteur. Cliquez sur ButtonEvent puis double-cliquez sur wx.Evt_BUTTON. Le terme « OnBtnAfficheNouveauButton » devrait apparaître un peu plus bas. Ensuite, sélectionnez le bouton « btnQuitter ». Faites la même chose en vérifiant que « OnBtnQuitterButton » apparaît bien. Envoyez et enregistrez. Ensuite, allez dans la fenêtre de l'éditeur et faites défiler jusqu'en bas. Assurez-vous que les deux méthodes

événements ont été créées. Voici à quoi devrait ressembler la fenêtre :



Maintenant, il est temps de nous occuper de l'autre fenêtre. Ouvrez DeuxiemeFenêtre avec l'éditeur graphique. Nommez-la « DeuxiemeFenetre » et paramétrez le titre à « Deuxième fenêtre ». Paramétrez le centrage à wx.BOTH. Ajoutez un wx.Button et centrez-le dans le bas de la fenêtre. Nommez-le « btnDFQuitter » et modifiez l'étiquette en « Quitter ». Ajoutez un événement à ce bouton. Ensuite, ajoutez un contrôle wx.StaticText dans la partie supérieure de la fenêtre, près du centre. Nommez-le « statSalut », paramétrez son étiquette à « Salut, je suis la deuxième fenêtre ! » et choisissez la police Sans, 14 points et la graisse (Weight) à wx.BOLD. Ensuite, recentrez horizontalement le message. Vous pouvez faire cela en décochant l'attribut position et en utilisant la position X pour la droite et la gauche et Y pour le haut et le bas jusqu'à ce que vous soyez satisfait. Envoyez et enregistrez .

Nous en avons dès lors terminé avec la mise en forme, nous allons créer la « glu » qui va les relier ensemble. Dans l'éditeur, cliquez sur



l'onglet GUI2 puis, en dessous, sur l'onglet Source. Sous la ligne qui dit « import FenetrePrincipale », ajoutez « import DeuxiemeFenetre ». Enregistrez les modifications. Ensuite sélectionnez l'onglet « FenetrePrincipale ». Sous la ligne qui dit « import wx », ajoutez la ligne « import DeuxiemeFenetre ». Descendez vers le bas et recherchez la ligne qui dit « def __init__(self, parent): ». Ajoutez la ligne « self.Fs = DeuxiemeFenetre.DeuxiemeFenetre(self) » après la ligne « self._init_ctrls(parent) ». Maintenant sous l'événement « def OnBtnAfficheNouveauButton(self, event): », commentez la ligne « event.Skip() » et ajoutez les deux lignes suivantes :

```
self.Fs.Show()
self.Hide()
```

Pour finir, sous la méthode « OnBtnQuitterButton », commentez la ligne « event.Skip() » et ajoutez la ligne « self.Close() ».

À quoi sert tout cela ? Eh bien, la première chose que nous avons faite, était de nous assurer que l'application savait que nous allions avoir deux fenêtres dans notre application. C'est pourquoi nous avons importé FenetrePrincipale et DeuxiemeFenetre dans le fichier GUI2. Ensuite, nous avons importé une référence à la DeuxiemeFenetre dans la FenetrePrincipale afin de pouvoir l'utiliser plus tard. Nous l'avons initialisée dans la méthode «___init___». Dans l'événement « OnBtnAfficheNouveauButton » nous lui avons dit que lors d'un clic sur le bouton, nous voulions d'abord afficher la seconde fenêtre puis cacher la fenêtre principale. Pour finir, nous avons saisi l'instruction pour fermer l'application lorsque le bouton Quitter est cliqué.

Maintenant, retournez au code de la DeuxiemeFenetre. Les modifications sont assez faibles. Sous la méthode «___init___», ajoutez une ligne qui dit « self.parent = parent » qui ajoute une variable self.parent. Finalement, sous l'événement associé au clic sur le bouton DFQuitterButton, commentez la ligne « event.Skip() » et ajoutez les deux lignes suivantes :

```
self.parent.Show()
self.Hide()
```

Souvenez-vous que nous avons caché la fenêtre principale lorsque nous avons affiché la deuxième fenêtre, nous devons donc l'afficher de nouveau. Enfin, nous cachons la seconde fenêtre. Enregistrez les modifications.

Voici le code complet, afin que vous puissiez tout vérifier (sur cette page et la suivante) : Maintenant vous pouvez lancer votre application. Si tout s'est bien passé, vous pouvez cliquer sur btnAfficheNouveau et voir la première fenêtre disparaître et la seconde fenêtre apparaître. Un clic sur le bouton Quitter sur la deuxième fenêtre la fait disparaître et la fenêtre principale réapparait. Un clic sur le bouton Quitter de cette fenêtre ferme l'application.

Je vous avais promis de vous parler des conventions pour donner des noms. Vous vous souvenez, nous avions parlé de mettre des commentaires dans votre code ? Eh bien, en utilisant des noms bien formés pour les contrôles de votre application, votre code sera presque auto-documenté.

Si vous laissez seulement des noms de contrôles tels que texteStatique1 ou bouton1 ou ce que vous voulez d'autre, lorsque vous êtes en train de créer une fenêtre complexe contenant beaucoup de contrôles, surtout s'il y a beaucoup de boîtes de texte ou de boutons, alors leur donner un nom qui est significatif est très important. Ce n'est peut-être pas si important si vous êtes le seul qui regardera le code mais pour quelqu'un qui passe derrière vous plus tard, les bon noms de contrôles l'aidera considérablement. Par conséquent, utilisez quelque chose comme cela :

```
Type de contrôle - Préfixe du nom
Texte statique - stat_
Bouton - btn_
Boîte de texte - txt_
```

GUI2 code:

```
#!/usr/bin/env python
#Boa:App:BoaApp

import wx

import FrameMain
import FrameSecond

modules ={u'FrameMain': [1, 'Main frame of Application',
u'FrameMain.py'],
 u'FrameSecond': [0, '', u'FrameSecond.py']}

class BoaApp(wx.App):
    def OnInit(self):
        self.main = FrameMain.create(None)
        self.main.Show()
        self.SetTopWindow(self.main)
        return True

def main():
    application = BoaApp(0)
    application.MainLoop()

if __name__ == '__main__':
    main()
```

FrameMain code:

```
#Boa:Frame:FrameMain

import wx
import FrameSecond

def create(parent):
    return FrameMain(parent)

[wxID_FRAMEMAIN, wxID_FRAMEMAINBTNEXIT,
wxID_FRAMEMAINBTNSHOWNEW,
 wxID_FRAMEMAINPANEL1,
] = [wx.NewId() for _init_ctrls in range(4)]

class FrameMain(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit
        wx.Frame.__init__(self, id=wxID_FRAMEMAIN,
name=u'FrameMain',
              parent=prnt, pos=wx.Point(846, 177),
size=wx.Size(400, 340),
              style=wx.DEFAULT_FRAME_STYLE, title=u'Main
Frame')
        self.SetClientSize(wx.Size(400, 340))
        self.Center(wx.BOTH)

        self.panel1 = wx.Panel(id=wxID_FRAMEMAINPANEL1,
name='panel1',
              parent=self, pos=wx.Point(0, 0),
size=wx.Size(400, 340),
              style=wx.TAB_TRAVERSAL)

        self.btnShowNew =
wx.Button(id=wxID_FRAMEMAINBTNSHOWNEW,
              label=u'Show the other frame',
name=u'btnShowNew',
              parent=self.panel1, pos=wx.Point(120, 103),
size=wx.Size(168, 29),
              style=0)
        self.btnShowNew.SetBackgroundColour(wx.Colour(25,
175, 23))
        self.btnShowNew.Bind(wx.EVT_BUTTON,
self.OnBtnShowNewButton,
              id=wxID_FRAMEMAINBTNSHOWNEW)
```

FrameMain Code (cont.):

```
        self.btnExit =
wx.Button(id=wxID_FRAMEMAINBTNEXIT, label=u'Exit',
              name=u'btnExit', parent=self.panel1,
pos=wx.Point(162, 191),
              size=wx.Size(85, 29), style=0)
        self.btnExit.SetBackgroundColour(wx.Colour(225,
218, 91))
        self.btnExit.Bind(wx.EVT_BUTTON,
self.OnBtnExitButton,
              id=wxID_FRAMEMAINBTNEXIT)

    def __init__(self, parent):
        self._init_ctrls(parent)
        self.Fs = FrameSecond.FrameSecond(self)

    def OnBtnShowNewButton(self, event):
        #event.Skip()
        self.Fs.Show()
        self.Hide()

    def OnBtnExitButton(self, event):
        #event.Skip()
        self.Close()
```

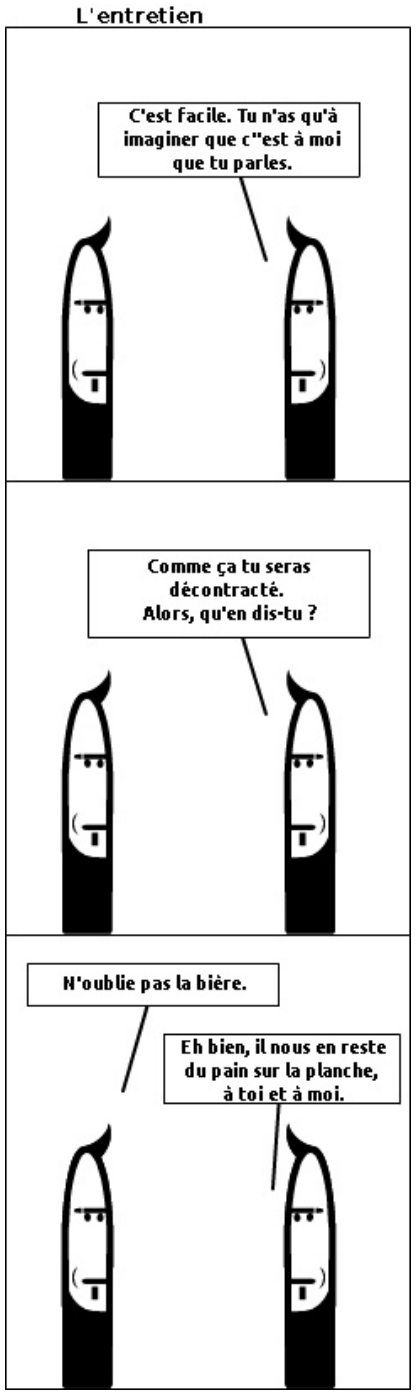
FrameSecond code:

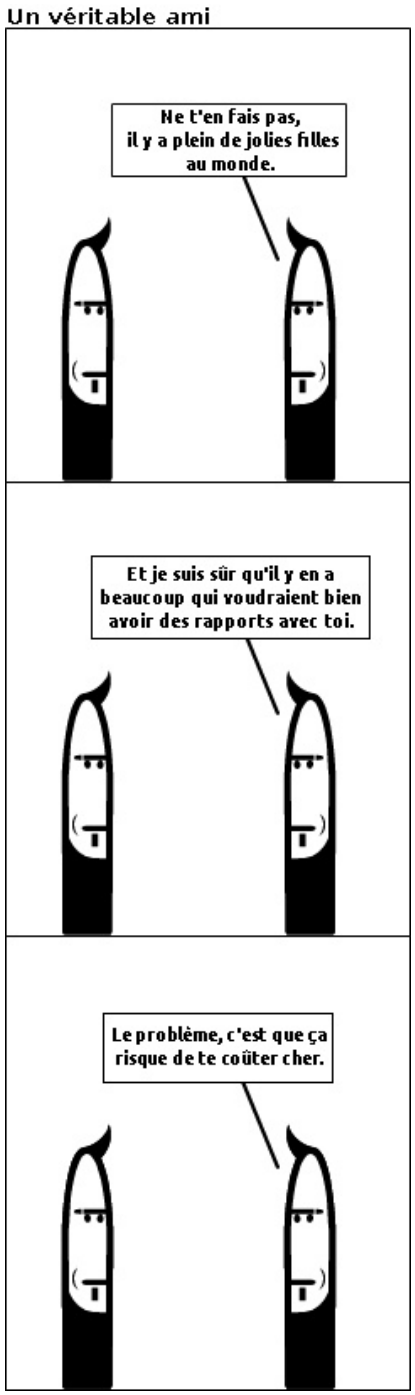
```
#Boa:Frame:FrameSecond

import wx

def create(parent):
    return FrameSecond(parent)

[wxID_FRAMESECOND, wxID_FRAMESECONDBTNFSEXIT,
wxID_FRAMESECONDPANEL1,
 wxID_FRAMESECONDSTATICTEXT1,
] = [wx.NewId() for _init_ctrls in range(4)]

class FrameSecond(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit
        wx.Frame.__init__(self, id=wxID_FRAMESECOND,
name=u'FrameSecond',
              parent=prnt, pos=wx.Point(849, 457),
size=wx.Size(419, 236),
              style=wx.DEFAULT_FRAME_STYLE,
title=u'Second Frame')
        self.SetClientSize(wx.Size(419, 236))
        self.Center(wx.BOTH)
        self.SetBackgroundStyle(wx.BG_STYLE_COLOUR)

        self.panel1 = wx.Panel(id=wxID_FRAMESECONDPANEL1,
name='panel1',
              parent=self, pos=wx.Point(0, 0),
size=wx.Size(419, 236),
              style=wx.TAB_TRAVERSAL)

        self.btnFSExit =
wx.Button(id=wxID_FRAMESECONDBTNFSEXIT, label=u'Exit',
              name=u'btnFSExit', parent=self.panel1,
pos=wx.Point(174, 180),
              size=wx.Size(85, 29), style=0)
        self.btnFSExit.Bind(wx.EVT_BUTTON,
self.OnBtnFSExitButton,
              id=wxID_FRAMESECONDBTNFSEXIT)

        self.staticText1 =
wx.StaticText(id=wxID_FRAMESECONDSTATICTEXT1,
              label=u"Hi there...I'm the second form!",
name='staticText1',
              parent=self.panel1, pos=wx.Point(45, 49),
size=wx.Size(336, 23),
              style=0)
        self.staticText1.SetFont(wx.Font(14, wx.SWISS,
wx.NORMAL, wx.BOLD,
              False, u'Sans'))

    def __init__(self, parent):
        self._init_ctrls(parent)
        self.parent = parent

    def OnBtnFSExitButton(self, event):
        #event.Skip()
        self.parent.Show()
        self.Hide()
```

```
Case à cocher - case_
Bouton radio - rad_
Cadre - cdr_ ou cadre_
```

Vous pourrez avoir vos propres idées de conventions pour les noms au fur et à mesure de vos progrès en tant que programmeur et, dans certains cas, il se peut que votre employeur ait déjà des conventions pré-établies.

La prochaine fois, nous laisserons un peu de côté la programmation des interfaces graphiques et nous nous concentrerons sur la programmation des bases de données. Entre temps, installez pythonapsw et python-mysqldb sur votre ordinateur. Vous aurez également besoin de sqlite et sqlitebrowser pour SQLite. Si vous voulez vous initier à MySql également, c'est une bonne idée. Tout cela est disponible via Synaptic.





TUTORIEL Programmer en Python - Partie 7

Greg Walters

Salut les gars et les filles. C'est l'heure du conte. Tout le monde est installé bien confortablement ? Prêts ? Allons-y !

Il était une fois un monde gouverné par le papier. Du papier, du papier partout. Il fallait construire des abris spéciaux pour stocker tout ce papier. On les appelait des classeurs à archives, et c'étaient de grosses choses en métal qui occupaient des pièces et des pièces et des pièces dans les bureaux pour stocker tout ce papier. Dans chaque classeur à archives, il y avait ce qu'on appelait un dossier plein de papiers, qui permettait d'essayer de regrouper les papiers en fonction de leur sujet. Mais au bout d'un moment, ils débordaient, ou se désagrégeaient quand ils devenaient trop vieux ou étaient consultés de trop nombreuses fois.

Pour utiliser ces classeurs à archives correctement, il fallait être diplômé. Trouver tous les papiers qui étaient rangés dans les différents classeurs pourrait prendre des jours. Les entreprises en souffraient beaucoup. Cette période de l'histoire de l'humanité fut extrêmement sombre.

Puis un jour, du haut d'une montagne quelque part (moi, je pense que c'était le Colorado, mais je n'en suis pas sûr), est arrivée une fée magnifique. Cette fée était bleue et argentée, avec de belles ailes et des cheveux blancs, et mesurait environ 30 centimètres. Son nom, croyez-le ou non, était Aiscuelle. N'est-ce pas un drôle de nom ? Peu importe, Aiscuelle a annoncé qu'elle pouvait régler tous les problèmes de papier, de classeurs à archives et de temps perdu, à condition que les gens croient en elle et en les ordinateurs. Son pouvoir se nommait « base de données ». Elle disait que les « bases de données » pouvaient remplacer tous les classeurs à archives existants. Certaines personnes l'ont cru et leur vie est rapidement devenue très heureuse. D'autres ne l'ont pas cru, et leur vie n'a pas changé, perdue au milieu de montagnes de papiers.

Cependant, toutes les promesses de fées sont soumises à des conditions. Cette fois-là, la condition était que, pour utiliser le pouvoir de Aiscuelle, il fallait apprendre à parler une nouvelle langue. Cela ne serait pas trop difficile à apprendre. En fait, cette langue ressemblait à celle que les gens utilisaient déjà. On disait simplement les choses un peu différemment, et il fallait bien, bien réfléchir avant de dire quelque chose pour utiliser le pouvoir de Aiscuelle.

Un jour, un jeune homme, curieusement appelé Utilisateur, vint voir Aiscuelle. Il était très impressionné par sa beauté, et lui demanda : « Aiscuelle, s'il te plaît, apprends-moi à utiliser ton pouvoir. » Aiscuelle lui répondit qu'elle allait le faire.

Elle lui dit : « D'abord, tu dois savoir comment ton information est organisée. Montre-moi tes papiers. »

Étant plutôt jeune, Utilisateur n'avait que quelques feuilles de papiers. Aiscuelle lui dit : « Utilisateur, pour l'instant tu pourrais vivre avec des papiers et des dossiers de fichiers. Mais je peux prédire l'avenir et, un jour, tu auras tant de papiers qu'en les empilant ils formeront un

Riz à l'espagnole

Pour 4 personnes

Source : Greg Walters

Ingrédients :
1 tasse de riz étuvé (cru)
500 g de bifteck haché
2 tasses d'eau
1 boîte de sauce tomate (250 g)
1 petit oignon émincé
1 gousse d'ail émincée
1 cuillère à soupe de cumin en poudre
1 cuillère à café d'origan en poudre
sel, poivre, sauce pimentée à volonté

Instructions :
Faire revenir la viande hachée dans une sauteuse.

Ajouter les autres ingrédients.

Porter à ébullition.

Remuer, couvrir,

puis laisser mijoter à feu doux pendant 20 minutes.

Ne pas regarder, ne pas toucher.

Remuer et servir.

tas 15 fois plus haut que toi. Nous devrions utiliser mon pouvoir. »

Et alors, en travaillant ensemble, Utilisateur et Aiscuelle créèrent cette chose appelée « base de données » (un terme technique de fée) et Utilisateur vécut heureux le restant de sa vie.

Fin.

Bien sûr, cette histoire n'est pas tout à fait vraie. Cependant, l'utilisation des bases de données et du langage SQL peut nous faciliter la vie. Nous allons maintenant apprendre quelques requêtes SQL simples et comment les utiliser dans un programme. Certains penseront qu'il ne s'agit pas d'une manière « correcte » ou de la « meilleure » manière de faire, mais c'est une manière raisonnable. Alors, allons-y.

Les bases de données sont comme les classeurs à archives de notre histoire. Et les tables sont comme les dossiers de fichiers. Chaque enregistrement distinct contenu dans les tables est comme une feuille de papier. Chaque renseignement est appelée un champ. Cela semble bien se goupiller, n'est-ce pas ? On utilise des requêtes SQL (prononcer Aiscuelle) pour manipuler les données. SQL signifie Structured Query Language (langage de requêtes structuré) et est conçu comme un moyen facile d'utiliser des bases de données. Mais en pratique, cela peut devenir très compliqué. Nous essaierons de rester simples dans cet épisode.

Nous devons commencer par créer un plan, comme dans tout projet de construction. Par exemple, pensez à une fiche de recette de cuisine ; c'est un bon exemple puisque nous allons créer une base de données de recettes. Chez moi, les recettes arrivent sous différentes formes : des fiches de format 3×5 pouces, des feuilles de papier 8×10 pouces, des serviettes où l'on a écrit une recette, des pages de magazines, et parfois des formes encore plus étranges. On peut les trouver dans des livres, des boîtes, des classeurs, etc. Cependant, elles ont toutes quelque chose en commun : leur format. Dans presque tous les cas, on trouve en haut le titre de la recette, et parfois le nombre de portions et la provenance de la recette. Au milieu, on trouve la liste des ingrédients et, en bas, les instructions à suivre pour préparer le plat, comme l'ordre dans lequel faire les choses, la durée de cuisson, etc. Nous utiliserons ce format général comme modèle pour notre base de données.

Nous allons découper le projet en deux parties : aujourd'hui nous allons nous occuper de la création de la base de données et, la prochaine fois, nous créerons l'application avec laquelle on peut consulter et mettre à jour les données.

Prenons un exemple. Supposons que nous ayons la recette indiquée (page précédente).

Remarquez l'ordre dont on vient de parler. Quand nous concevrons la base de données, on pourrait envisager de stocker toutes les informations de la recette dans un seul gros enregistrement. Mais ce serait lourd et difficile à gérer ensuite. Au lieu de cela, nous allons utiliser la fiche de la recette comme un modèle. Une table stockera la partie du haut, c'est-à-dire les informations générales de la recette ; une autre table s'occupera du milieu, les ingrédients, et nous aurons une dernière table pour le bas, les instructions.

Assurez-vous d'avoir installé SQLite et APSW. SQLite est un petit gestionnaire de bases de données qui fonctionne sans avoir besoin d'installer un serveur de bases de données, ce qui est parfait pour notre petite application. Tout ce que vous allez apprendre ici peut être utilisé sur de plus gros systèmes de gestion de bases de données comme MySQL ou d'autres. L'autre qualité de SQLite est qu'il utilise des types de données en nombre limité. Ces types sont Text, Numeric, Blob et Integer Primary Keys. Comme on l'a déjà vu, le type Text permet de stocker diverses informations textuelles. Les ingrédients, les instructions, le titre de la recette sont tous de type Text, même s'ils contiennent parfois des nombres. Le type Numeric permet de stocker des nombres, qui peuvent être des entiers ou des nombres réels ou à virgule flottante. Le type Blob permet de stocker des données binaires, comme par exemple des images. Integer Primary Key (clé primaire entière) est un peu spécial ; SQLite s'en sert pour y enregistrer automatiquement un nombre entier unique. Ceci est important pour la suite.

APSW signifie Another Python SQLite Wrapper (un autre intermédiaire Python à SQLite) et permet de communiquer facilement avec SQLite. Maintenant, voyons différentes façons de créer des requêtes SQL.

Pour retrouver les enregistrements d'une base de données, on utilise une instruction SELECT, dont la syntaxe est la suivante :

```
SELECT [quoi] FROM
[quelle(s) table(s)] WHERE
[des contraintes]
```

Ainsi, pour retrouver tous les champs de la table Recettes, on écrira :

```
SELECT * FROM Recettes
```

Si on ne souhaite obtenir qu'un seul enregistrement à partir de sa clé primaire, on doit connaître la valeur de cette clé (pkID dans notre exemple) et ajouter la commande WHERE, par exemple, ainsi :

```
SELECT * FROM Recettes WHERE
pkID = 2
```

Plutôt simple, n'est-ce pas ? Presque du langage courant. Maintenant, supposons qu'on veuille juste obtenir le nom de la recette et le nombre de portions, et ceci pour toutes les recettes. C'est facile. Tout ce qu'on a à faire est d'inclure une liste de champs dans la requête SELECT :

```
SELECT nom, portions FROM
Recettes
```

Pour insérer des enregistrements, on utilise la commande INSERT INTO. La syntaxe est :

```
INSERT INTO [nom de table]
(liste des champs) VALUES
(valeurs à insérer)
```

Par exemple, pour insérer une recette dans la table des recettes, la commande serait :

```
INSERT INTO Recettes
(nom,portions,source) VALUES
("Tacos",4,"Greg")
```

Pour effacer un enregistrement, on utilise :

```
DELETE FROM Recettes WHERE
pkID = 10
```

Il y a aussi une instruction UPDATE, mais nous verrons cela plus tard.

Plus d'informations sur SELECT.

Dans le cas de notre base de données, nous avons trois tables, reliées entre elles grâce à idRecette qui pointe vers pkID de la table Recettes. Disons que l'on veut récupérer les instructions d'une recette donnée. On peut le faire ainsi :

```
SELECT Recettes.nom,
Recettes.personnes,
Recettes.source,
Instructions.Instructions
FROM Recettes LEFT JOIN
instructions ON
(Recettes.pkid =
Instructions.idRecette)
WHERE Recettes.pkid = 1
```

Cependant, ceci est long à taper et très redondant. On peut utiliser des alias ainsi :

```
SELECT  r.nom, r.personnes,
r.source, i.Instructions
FROM Recettes r LEFT JOIN
instructions i ON (r.pkid =
i.idRecette) WHERE r.pkid = 1
```

C'est plus court et ça reste lisible. Écrivons maintenant un petit programme qui va créer notre base de données, les tables, et y entrer quelques données, afin que nous puissions ensuite travailler avec. Nous POURRIONS écrire tout ça dans notre futur programme complet, mais, dans cet exemple, nous écrirons un programme séparé. Celui-ci sera un programme à utilisation unique ; si vous essayez de l'exécuter une deuxième fois, il échouera à la création des tables. Encore une fois, nous pourrions insérer le code dans une instruction « try…catch » pour éviter le plantage, mais nous ferons cela une autre fois.

Commençons par importer l'adaptateur APSW :

```
import apsw
```

L'étape suivante consiste à créer une connection à notre base de données. Elle sera placée dans le même répertoire que notre application. Lorsqu'on crée cette connexion, SQLite vérifie automatiquement que la base existe. Si c'est le cas, elle est ouverte. Sinon, la base est créée pour nous. Une fois la connexion établie, nous aurons besoin de quelque chose qui s'appelle un curseur. Celui-ci crée un mécanisme qu'on utilise pour travailler avec la base de données. Pour résumer, nous avons donc besoin d'une connexion et d'un curseur. Voici comment les créer :

```
connexion=apsw.Connection("l
ivrerecettes1.db3")
curseur=connexion.curseur()
```

Bon, nous avons une connexion et un curseur. Il faut maintenant créer des tables. Il y aura trois tables dans notre application. L'une contiendra les informations générales de la recette, une autre, les instructions pour chaque recette, et une dernière, la liste des ingrédients. N'aurions-nous pas pu faire cela avec une seule table ? Si, bien sûr, mais comme vous le verrez, cela ferait une table vraiment grosse, et il y aurait beaucoup d'informations dupliquées.

Considérons la structure de tables suivante : chaque colonne représente une table comme ci-après :

```
RECETTES                    INSTRUCTIONS                  INGRÉDIENTS
------------                ----------------------        --------------------
pkID (Integer Primary Key)  pkID(Integer Primary Key)     pkID (Integer Primary Key)
nom (Text)                  idRecette (Integer)           idRecette (Integer)
source (Text)               instructions (Text)           ingrédients (Text)
Nbpersonnes (Text)
```

Chaque table possède un champ nommé pkID. C'est la clé primaire, qui sera unique à l'intérieur de la table. C'est important, afin que les tables de données ne contiennent jamais un enregistrement qui soit complètement identique à un autre. Cet identificateur est un entier qui est attribué automatiquement par le moteur de base de données. Pourrait-on se passer de l'attribution automatique ? Oui, mais en courant le risque de créer accidentellement un identificateur d'enregistrement en double. Dans le cas de la table Recettes, nous utiliserons ce nombre pour retrouver quelles instructions et quels ingrédients vont avec telle ou telle recette.

Nous allons d'abord saisir les informations de nom, source et Nbpersonnes dans la table Recettes. pkID est attribué automatiquement. Lorsqu'on insère le tout premier enregistrement dans la table, il obtiendra un pkID de 1. Nous utiliserons cette valeur pour relier l'information des autres tables à cette recette. La table Instructions est simple. Elle contient simplement un long texte contenant les instructions, son propre pkID, et un pointeur vers la recette de la table Recettes. La table Ingrédients est un peu plus complexe puisqu'elle contient un enregistrement pour chaque ingrédient, ainsi que son propre pkID et un pointeur vers un enregistrement de la table Recettes.

Ainsi, pour créer la table Recettes, on définit une chaîne de caractères dans une variable appelée sql, et on y place la commande pour créer la table :

```
sql = 'CREATE TABLE Recettes
(pkID INTEGER PRIMARY KEY,
nom TEXT, Nbpersonnes TEXT,
source TEXT)
```

Puis on demande à APSW d'exécuter cette commande :

```
curseur.execute(sql)
```

Enfin, on crée les autres tables :

```
sql = 'CREATE TABLE
Instructions (pkID INTEGER
PRIMARY KEY, instructions
TEXT, idRecette NUMERIC)'

curseur.execute(sql)

sql = 'CREATE TABLE
Ingredients (pkID INTEGER
PRIMARY KEY, ingredients
TEXT, idRecette NUMERIC)'

curseur.execute(sql)
```

Une fois les tables créées, on utilise l'instruction INSERT INTO pour entrer chaque ensemble de données dans la table appropriée.

Souvenez-vous que pkID est automatiquement attribué, on n'en tient donc pas compte dans la liste des champs de la commande INSERT. Et comme on précise le nom des champs, on peut les mettre dans n'importe quel ordre, pas forcément l'ordre dans lequel ils ont été placés à la création de la table. Dès lors que nous connaissons le nom des champs, tout fonctionnera parfaitement. Voici l'instruction INSERT pour la table Recettes :

```
INSERT INTO Recettes (nom,
Nbpersonnes, source) VALUES
(“Riz à l'espagnole”,4,”Greg
Walters”)
```

Ensuite nous devons récupérer la valeur affectée à pkID dans la table Recettes. On peut faire ça avec une simple commande :

```
SELECT last_insert_rowid()
```

Cependant, ce n'est pas tout à fait aussi simple, il faut plutôt une suite d'instructions comme celle-ci :

```
sql = "SELECT
last_insert_rowid()"

curseur.execute(sql)

for x in
curseur.execute(sql):
    dernierID = x[0]
```

Pourquoi ? Que veut dire tout cela ? En fait, lorsque APSW nous renvoie des données, celles-ci nous arrivent sous forme de tuple. Nous n'avons pas encore parlé de cela. En deux mots, un tuple ressemble à une liste, mais qui n'est pas modifiable. Certains utilisent rarement les tuples, d'autres les utilisent souvent ; c'est un choix. Ce qui importe c'est que nous voulons utiliser la première

des valeurs renvoyées. On utilise une boucle « for » pour récupérer la première valeur du tuple x. C'est compris ? Bon, continuons.

L'étape suivante consiste à créer la requête d'insertion pour les instructions :

```
sql = 'INSERT INTO
Instructions
(idRecette,instructions)
VALUES( %s,"Faire revenir la
viande hachée dans une
sauteuse. Ajouter les autres
ingrédients. Porter à
ébullition. Remuer, couvrir,
puis laisser mijoter à feu
doux pendant 20 minutes. Ne
regardez pas, ne touchez
pas. Remuer et servir.")' %
dernierID

curseur.execute(sql)
```

Notez que l'on utilise la substitution de variable (%s) pour placer le pkID de la recette (dernierID) dans la requête SQL. Enfin, nous devons placer chaque ingrédient dans la table Ingredients. En voici un exemple :

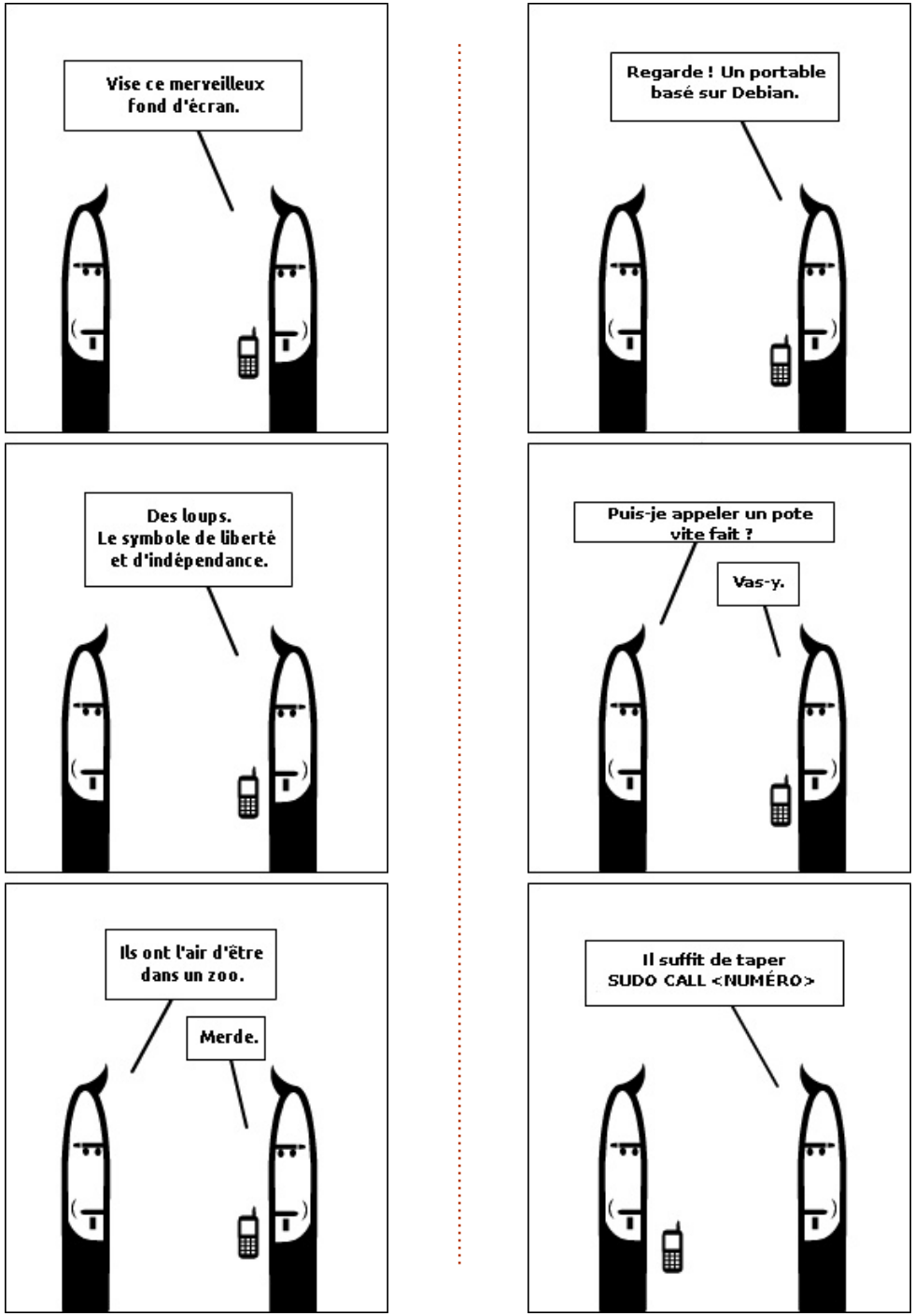
```
sql = 'INSERT INTO
Ingredients
(idRecette,ingredients)
VALUES ( %s,"1 tasse de riz
étuvé (cru)")' % dernierID
curseur.execute(sql)

curseur.execute(sql)
```

Ce n'est pas dur à comprendre jusque-là. La prochaine fois, ce sera un peu plus compliqué.

Si vous voulez récupérer le code source, je l'ai mis sur mon site Web ; vous pouvez le télécharger ici : www.thedesignatedgeek.com.

La prochaine fois, nous utiliserons ce que nous avons appris depuis le début de cette série pour créer une interface avec des menus pour accéder à nos recettes. Cette interface permettra de voir la liste des recettes, de consulter le détail de chaque recette, de rechercher une recette, et d'ajouter ou de supprimer des recettes.

Je vous suggère de passer un peu de temps à lire des documents sur la programmation SQL. Vous vous en féliciterez la prochaine fois.



TUTORIEL Programmer en Python - Partie 8

Greg Walters

Continuons la programmation de notre base de données de recettes que nous avons commencée dans la partie 7. Cela sera un peu plus long avec beaucoup de lignes de code donc accrochez-vous, ne lâchez pas prise et gardez les mains sur le volant. Nous avons déjà créé notre base de données, maintenant nous voulons en afficher le contenu et y ajouter et supprimer des éléments. Comment procède-t-on ? Nous commencerons par une application qui se lance dans un terminal, nous devons donc créer un menu. Nous allons également créer une classe qui contiendra nos sous-programmes pour la base de données.

Commençons par l'échantillon de programme affiché en haut à droite. Commençons par mettre en forme notre menu pour nous permettre de mettre en place notre classe. Notre menu sera une grande boucle qui affichera une liste d'options que l'utilisateur pourra choisir. Nous utilisons une boucle « while ». Modifiez la procédure menu afin qu'elle ressemble à celle affichée en bas à droite.

Ensuite, nous complétons le menu par la structure if|elif|else qui est affichée en haut de la page suivante.

Jetons un coup d'œil à notre sous-programme menu. Nous commençons par afficher les actions que l'utilisateur peut faire. Nous initialisons une variable (boucle) à True, puis nous utilisons la structure « while » pour continuer la boucle jusqu'à ce que boucle = False. Nous utilisons la commande raw_input() pour attendre le choix de l'utilisateur et enfin notre structure « if » gère l'option que l'utilisateur a choisie. Avant de pouvoir tester cela, nous devons créer une ébauche du sous-programme __init__ dans notre classe :

```
def __init__(self):
pass
```

Enregistrez votre programme là où vous avez enregistré la base de données que vous avez créée la dernière fois et lancez-le. Vous devriez voir quelque chose comme ce qui est affiché sur la page suivante, en haut, à droite.

Cela devrait tout simplement afficher le menu encore et encore jusqu'à ce que vous tapiez « 0 », puis affichiez « Au revoir » et quittiez.

```
#!/usr/bin/python
#-----------------------------------------------------
# LivreDeRecettes.py
# Créé pour Programmation Python partie 8
# et Full Circle Magazine
#-----------------------------------------------------
import apsw
import string
import webbrowser

class LivreDeRecettes:

def Menu():
cbk = LivreDeRecettes() # Initialise la classe
Menu()
```

```
def Menu():
    cbk = LivreDeRecettes() # Initialise la classe
    boucle = True
    while boucle == True:
        print
'=================================================='
        print          'BASE DE DONNEES DE RECETTES'
        print
'=================================================='
        print ' 1 - Afficher toutes les recettes'
        print ' 2 - Chercher une recette'
        print ' 3 - Afficher une recette'
        print ' 4 - Supprimer une recette'
        print ' 5 - Ajouter une recette'
        print ' 6 - Imprimer une recette'
        print ' 0 - Quitter'
        print
'=================================================='
        reponse = raw_input('Saisissez votre choix -> ')
```

```
if reponse == '1': # Affiche toutes les recettes
     pass
elif reponse == '2': # Recherche une recette
     pass
elif reponse == '3': # Affiche une seule recette
     pass
elif reponse == '4': # Supprime une recette
     pass
elif reponse == '5': # Ajoute une recette
     pass
elif reponse == '6': # Imprime une recette
     pass
elif reponse == '0': # Quitte le programme
     print 'Au revoir'
     boucle = False
   else:
     print 'Commande inconnue. Essayez encore.'
```

Nous pouvons maintenant compléter nos routines de la classe LivreDeRecettes. Nous avons besoin d'une routine qui affiche toutes les informations de la table de données Recettes, d'une qui vous permet de rechercher une recette, d'une qui vous affiche les données d'une seule recette contenues dans les trois tables, d'une qui supprime une recette, d'une qui permet d'ajouter une recette et d'une qui imprime la recette sur l'imprimante par défaut. La routine AfficheTout n'a besoin que d'un seul paramètre (self) tout comme les sous-programmes Cherche1Recette ou SaisirNouvelle. Les routines Affiche1Recette, Supprime1Recette et ImprimeTout ont toutes besoin de savoir à quelle recette vous faites référence, elles nécessitent donc un paramètre que nous appellerons « laquelle ». Utilisez la commande « pass » pour terminer chaque cas. Dans la classe LivreDeRecettes, créez les éléments :

```
def AfficheTout(self):
  pass
def Cherche1Recette(self):
  pass
def Affiche1Recette(self,laquelle):
  pass
def Supprime1Recette(self,laquelle):
  pass
def SaisirNouvelle(self):
  pass
def ImprimeTout(self,laquelle):
  pass
```

```
/usr/bin/python -u
 "/home/greg/python_examples/APSW/cookbook/cookbook_stub.py"
==================================================
BASE DE DONNEES DE RECETTES
==================================================
1 - Afficher toutes les recettes
2 - Chercher une recette
3 - Afficher une recette
4 - Supprimer une recette
5 - Ajouter une recette
6 - Imprimer une recette
0 - Quitter
==================================================
Saisissez votre choix ->
```

Dans un grand nombre de cas du menu, nous devons afficher toutes les recettes de la table Recette afin que l'utilisateur puisse choisir ce qui l'intéresse dans cette liste. Cela concerne les options 1, 3, 4 et 6. Il faut donc modifier la routine du menu pour ces options en remplaçant la commande pass par cbk.AfficheTout().

Notre routine de vérification de la réponse ressemblera maintenant au code en haut de la page suivante.

Il reste à écrire la routine __init__. Remplacez l'ébauche par les lignes suivantes :

```
def __init__(self):
  global connexion
  global curseur
  self.nombretotal = 0
  connexion=apsw.Connection(
"livrerecettes.db3")
  curseur=connexion.cursor()
```

Nous commençons par créer deux variables globales pour notre connexion et notre curseur. Nous pouvons y accéder à partir de n'importe quel endroit de la classe LivreDeRecettes. Ensuite, nous créons une variable self.nombretotal que nous utiliserons plus tard pour compter le nombre de recettes. Enfin nous créons la connexion et le curseur. L'étape suivante est de compléter la routine cbk.AfficheTout dans la classe LivreDeRecettes.

Puisque les variables pour la connexion et le curseur sont globales,

```
if response == '1': # Affiche toutes les recettes
     cbk.AfficheTout()
elif response == '2': # Recherche une recette
     pass
elif response == '3': # Affiche une seule recette
     cbk.AfficheTout()
elif response == '4': # Supprime une recette
     cbk.AfficheTout()
elif response == '5': # Ajoute une recette
     pass
elif response == '6': # Imprime une recette
     cbk.AfficheTout()
elif response == '0': # Quitte le programme
     print 'Au revoir'
     boucle = False
else:
     print 'Commande inconnue. Essayez encore.'
```

nous n'avons pas à les créer à nouveau dans chaque routine. Nous voulons un affichage « sympa » à l'écran des en-têtes pour notre liste de recettes. Nous utiliserons la commande de formatage « %s » et celle de justification à gauche pour espacer la sortie à l'écran. Nous voulons qu'elle ressemble à ceci :

```
Numéro Nom Personnes Source
--------------------------
```

Enfin nous devons fabriquer notre instruction SQL, envoyer la requête à la base de données et afficher les résultats. La plupart des choses ont été vues dans l'article précédent.

```
   sql = 'SELECT * FROM
Recettes'
   cntr = 0
   for x in
curseur.execute(sql):
      cntr += 1
      print '%s %s %s %s'
%(str(x[0]).rjust(5),x[1].l
just(30),x[2].ljust(20),x[3
].ljust(30))
   print '-------------'
   self.nombretotal = cntr
```

La variable cntr permet de compter le nombre de recettes que l'utilisateur voit. La routine est terminée. Le code complet est affiché à droite au cas où vous ayez raté quelque chose.

Remarquez que nous utilisons le tuple qui est renvoyé par la routine curseur.execute de ASPW. Nous affichons le pkID comme numéro de recette, cela nous permettra de choisir la bonne recette plus tard. Lorsque vous lancez le programme, le menu s'affiche et si vous choisissez l'option 1, vous obtenez ce qui est affiché en haut de la page suivante. C'est ce que nous voulions, sauf si vous lancez l'application avec Dr.Python ou quelque chose du même style auquel cas le programme ne fait pas de pause. Ajoutons une pause qui attend que l'utilisateur appuie sur une touche afin d'avoir le temps de regarder ce qui s'affiche. Pendant que nous y sommes, affichons le nombre total de recettes à l'aide de la variable paramétrée tout à l'heure. Ajoutez en bas de l'option 1 du menu :

```
print 'Nombre de recettes -
 %s' %cbk.nombretotal
print '--------------------
-------------------------
-----'
res = raw_input('Appuyez
sur une touche -> ')
```

Oublions l'option 2 pour l'instant (recherche d'une recette) et parlons de l'option 3 (afficher une seule recette). Intéressons-nous d'abord à la partie menu.

Nous affichons la liste des recettes, comme pour l'option 1, puis nous demandons à l'utilisateur d'en choisir une. Pour être sûr qu'il n'y ait pas d'erreur à cause d'une mauvaise saisie de l'utilisateur, nous utilisons une structure Try|Except. Nous affichons le message (Choisissez une recette □), puis si la réponse est correcte, nous appelons la routine Affiche1Recette()

```
def AfficheTout(self):
        print '%s %s %s %s'
%('Numéro'.ljust(5),'Nom'.ljust(30),'Personnes'.ljust(
20),'Source'.ljust(30))
        print '--------------------------------'
        sql = 'SELECT * FROM Recipes'
        cntr = 0
        for x in curseur.execute(sql):
        cntr += 1
        print '%s %s %s %s'
%(str(x[0]).rjust(5),x[1].ljust(30),x[2].ljust(20),x[3
].ljust(30))
        print '--------------------------------'
        self.nombretotal = cntr
```

```
Saisissez votre choix -> 1
Numéro Nom                          Personnes            Source
------------------------------------------------------------------------------
     1 Riz à l'espagnole              4                  Greg
     2 Poivrons et oignons marinés  9 bocaux             Le guide complet des conserves
------------------------------------------------------------------------------
================================================
BASE DE DONNEES DE RECETTES
================================================
1 - Afficher toutes les recettes
2 - Chercher une recette
3 - Afficher une recette
4 - Supprimer une recette
5 - Ajouter une recette
6 - Imprimer une recette
0 - Quitter
================================================
Saisissez votre choix ->
```

dans notre classe LivreDeRecettes avec le pkID de notre table Recette. Si l'entrée n'est pas un nombre, cela créera une exception ValueError que nous gérons avec l'instruction except ValueError (copie d'écran à droite).

Ensuite, nous travaillons routine Affiche1Recette dans notre classe LivreDeRecettes. Commençons par la connexion et le curseur à nouveau, puis créons notre instruction SQL. Dans ce cas, nous utilisons « SELECT * FROM Recettes WHERE pkID = %s" % str(laquelle) » où laquelle est la valeur que nous voulons obtenir. Ensuite nous fabriquons un bel affichage toujours à l'aide du tuple renvoyé par ASPW. Dans ce cas, nous utilisons x comme variable brute, puis chaque élément avec l'index entre crochets dans le tuple. Puisque l'agencement de la table est pkID / nom / NBpersonnes / source, nous pouvons utiliser x[0], x[1], x[2] et x[3] pour le détail. Ensuite, nous voulons récupérer le contenu de la table Ingrédients dont le idRecette (notre clé dans la table des données Recettes) est égal au pkID que nous venons d'utiliser. Nous parcourons le tuple renvoyé, affichant chaque ingrédient puis nous obtenons finalement les instructions de la table Instructions, comme nous l'avons fait pour la table Ingrédients. Enfin, nous attendons que l'utilisateur appuie sur une touche afin qu'il puisse lire la recette à l'écran. Le code est donné sur la page suivante.

Maintenant, nous avons deux routines de terminées sur les six. Parlons de la routine de recherche en commençant à nouveau par le menu. Heureusement cette fois-ci, nous ne faisons qu'appeler la routine de recherche de la classe donc remplacez la commande pass par :

```
cbk.Cherche1Recette()
```

Maintenant complétons notre code de recherche. Dans la classe LivreDeRecettes, remplacez l'ébauche de Cherche1Recette par le code affiché à la page 39.

Beaucoup de choses se passent. Après la création de notre connexion et curseur, nous affichons notre menu de recherche. Nous proposons 3 méthodes de recherche à l'utilisateur et un moyen de quitter la routine. Il est possible de chercher un mot dans le nom de la recette, un mot dans la source de la recette ou un mot dans la liste des ingrédients. À cause de cela, nous ne pouvons pas utiliser la routine d'affichage que nous venons

```
try:
    res = int(raw_input('Choisissez une recette -> '))
    if res <= cbk.nombretotal:
    cbk.Affiche1Recette(res)
    elif res == cbk.nombretotal + 1:
    print 'Retour au menu...'
    else:
    print 'Commande inconnue. Retour au menu.'
except ValueError:
    print "Ce n'est pas un nombre... Retour au menu."
```

de créer et nous devons créer des sous-programmes de sorties personnalisées. Les deux premières options utilisent des instructions SELECT simples avec une petite astuce. Nous utilisons le qualificatif « like ». Si nous utilisions un logiciel comme SQLite Database Browser, notre instruction like utiliserait un caractère joker « % ». Donc pour rechercher une recette contenant le mot « riz » dans le nom de la recette, notre requête serait :

```
SELECT * FROM Recettes
WHERE nom like '%riz%'
```

Cependant, comme le caractère « % » est également un caractère de substitution dans nos chaînes de caractères, nous devons utiliser %% dans notre texte. Pour compliquer la chose, nous utilisons le caractère de substitution pour insérer le mot que l'utilisateur recherche. Par conséquent, nous devons le transformer comme cela '%%%s%%'. Désolé, si ce n'est pas très clair. La troisième requête est appelée une instruction Join. Regardons-la d'un peu plus près :

```
sql = "SELECT
r.pkid,r.nom,r.NBpersonnes,r

.source,i.ingredients FROM
Recettes r Left Join
ingredients i on (r.pkid =
i.idRecette) WHERE
i.ingredients like '%%%s%%'
GROUP BY r.pkid" %response
```

Nous sélectionnons tout dans la table Recette et les ingrédients dans la table Ingrédients, en utilisant un « join » pour créer une relation entre les éléments de la table Ingrédients et ceux la table Recette de façon que idRecette soit égal à pkID, puis en recherchant notre ingrédient grâce à l'instruction like. Finalement, nous regroupons le résultat par pkID dans la table Recettes pour éviter que des doublons soient affichés. Si vous vous souvenez, nous avons des poivrons deux fois dans la seconde recette (poivrons et oignons marinés), un vert et un rouge. Cela pourrait prêter à confusion dans l'esprit de notre utilisateur.

Notre menu utilise :

```
  searchin =
raw_input('Saisissez le
type de recherche -> ')
if searchin != '4':
```

qui indique : si searchin (la valeur que l'utilisateur saisit) n'est PAS égale à 4 alors s'occuper des options, si c'est 4 alors ne rien faire et continuer à la suite. Notez que j'ai utilisé « != » pour « différent de » au lieu de « <> ». Les deux fonctionneraient sous Python 2.x. Cependant, en Python 3.x, cela serait une erreur de syntaxe. Nous parlerons plus des modifications Python 3.x dans un futur article. Utilisez « != » dès maintenant pour vous faciliter la vie pour migrer vers Python 3.x plus tard. Enfin, nous fabriquons encore un « bel affichage ». Regardons ce que verra l'utilisateur, affiché page 40.

```
def Affiche1Recette(self,laquelle):
sql = 'SELECT * FROM Recettes WHERE pkID = %s' %
str(laquelle)
print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
for x in curseur.execute(sql):
idrecette =x[0]
print "Titre : " + x[1]
print "NbPersonnes : " + x[2]
print "Source : " + x[3]
print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
sql = 'SELECT * FROM Ingredients WHERE idRecette =
%s' % idrecette
print 'Liste des ingredients :'
for x in curseur.execute(sql):
print x[1]
print ''
print 'Instructions :'
sql = 'SELECT * FROM Instructions WHERE idRecette
= %s' % idrecette
for x in curseur.execute(sql):
print x[1]
print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
resp = raw_input('Appuyez sur une touche -> ')
```

Vous pouvez voir comme la sortie du programme est belle. Maintenant, l'utilisateur peut retourner au menu et utiliser l'option 3 pour afficher la recette qu'il veut. Ensuite, nous voulons ajouter des recettes dans notre base de données. À nouveau, nous devons juste ajouter une ligne dans la routine de menu pour appeler la routine SaisirNouvelle :

```
cbk.SaisirNouvelle()
```

```
def Cherche1Recette(self):
      # Affiche le menu de recherche
      print '------------------------------'
      print ' Recherche dans'
      print '------------------------------'
      print ' 1 - Nom de la recette'
      print ' 2 - Source de la recette'
      print ' 3 - Ingrédients'
      print ' 4 - Quitter'
      searchin = raw_input('Saisissez le type de recherche -> ')
      if searchin != '4':
          if searchin == '1':
               search = 'Nom de la recette'
          elif searchin == '2':
               search = 'Source de la recette'
          elif searchin == '3':
               search = 'Ingrédients'
          parm = searchin
          response = raw_input('Recherche dans : %s (blanc pour quitter) -> ' % search)
          if parm == '1': # Nom de la recette
               sql = "SELECT pkid,nom,source,NBpersonnes FROM Recettes WHERE nom like '%%%s%%'" %response
          elif parm == '2': # Source de la recette
               sql = "SELECT pkid,nom,source,NBpersonnes FROM Recettes WHERE source like '%%%s%%'" %response
          elif parm == '3': # Ingredients
               sql = "SELECT r.pkid,r.nom,r.NBpersonnes,r.source,i.ingredients FROM Recettes r Left Join ingredients i
on (r.pkid = i.idRecette) WHERE i.ingredients like '%%%s%%' GROUP BY r.pkid" %response
          try:
               if parm == '3':
                 print '%s %s %s %s %s'
%('Numéro'.ljust(5),'Nom'.ljust(30),'Personnes'.ljust(20),'Source'.ljust(30),'Ingrédient'.ljust(30))
                 print '--------------------------------------------------------------------------------'
               else:
                 print '%s %s %s %s' %('Numéro'.ljust(5),'Nom'.ljust(30),'Personnes'.ljust(20),'Source'.ljust(30))
                 print '-------------------------------------------------------------------------------'
               for x in curseur.execute(sql):
                 if parm == '3':
                    print '%s %s %s %s %s'
%(str(x[0]).rjust(5),x[1].ljust(30),x[2].ljust(20),x[3].ljust(30),x[4].ljust(30))
                 else:
                    print '%s %s %s %s' %(str(x[0]).rjust(5),x[1].ljust(30),x[3].ljust(20),x[2].ljust(30))
          except:
               print 'Il y a une erreur'
          print '-------------------------------------------------------------------------------'
          inkey = raw_input('Appuyez sur une touche')
```

Le code de SaisirNouvelle(), qui doit remplacer l'ébauche dans la classe LivreDeRecettes, se trouve à : http://pastebin.com/Ce0fMphZ.

Nous commençons par définir une liste appelée « ings » comme ingrédients. Puis, nous demandons à l'utilisateur de saisir le titre, la source et le nombre de personnes. Puis nous entrons dans une boucle qui demande chaque ingrédient en l'ajoutant à la liste ings. Si l'utilisateur saisit 0, nous quittons la boucle et nous continuons en demandant des instructions. Nous affichons alors le contenu de la recette et demandons à l'utilisateur de vérifier avant d'enregistrer les données. Nous utilisons les instructions INSERT INTO, comme la dernière fois, et retournons au menu. Il faut faire attention aux apostrophes simples dans nos entrées. NORMALEMENT, cela n'est pas un problème dans la liste des ingrédients ou les instructions, mais dans nos champs titre et source, cela pourrait l'être. Nous devons ajouter un caractère d'échappement à chaque apostrophe simple. Nous faisons cela avec la routine string.replace, c'est pourquoi nous avons importé la bibliothèque string. Dans la routine du menu, mettez le code affiché sur la droite sous l'option 4. Puis, dans la classe

```
Saisissez votre choix -> 2
------------------------------
Recherche dans
------------------------------
1 - Nom de la recette
2 - Source de la recette
3 - Ingrédients
4 - Quitter
Saisissez le type de recherche -> 1
Recherche dans : Nom de la recette (blanc pour quitter) -> riz
Numéro  Nom                    Personnes               Source
-------------------------------------------------------------------------------
     1  Riz à l'espagnole          4                    Greg
-------------------------------------------------------------------------------
Appuyez sur une touche
```

Assez simple. Maintenant pour la recherche d'ingrédients...

```
Saisissez votre choix -> 2
------------------------------
Recherche dans
------------------------------
1 - Nom de la recette
2 - Source de la recette
3 - Ingrédients
4 - Quitter
Saisissez le type de recherche -> 3
Recherche dans : Ingrédients (blanc pour quitter) -> oignon
Numéro  Nom                  Personnes      Source                    Ingrédient
-------------------------------------------------------------------------------
     1  Riz à l'espagnole        4            Greg                1 petit oignon émincé
     2  Poivrons et oignons   9 bocaux    Le guide complet         6 tasses d'oignons ciselés
        marinés                            des conserves

-------------------------------------------------------------------------------
Appuyez sur une touche
```

LivreDeRecettes, utilisez le code affiché ci-contre en bas pour la routine Supprime1Recette().

Parcourons rapidement la routine de suppression. Nous demandons tout d'abord la recette à supprimer (retour au menu) et transmettons le numéro pkID à notre routine de suppression. Ensuite nous demandons confirmation à l'utilisateur. Si la réponse est « O » (string.upper(resp) == 'O') alors nous créons nos requêtes de suppression sql. Notez que, cette fois-ci, nous devons supprimer des entrées dans les trois tables. Nous aurions pu très certainement supprimer seulement l'entrée de la table Recettes, mais nous aurions alors des entrées orphelines dans les deux autres et ce n'est pas très bien. Lorsque nous supprimons l'entrée de la table Recettes, nous utilisons le champ pkID. Dans les deux autres tables, nous utilisons le champ idRrecette.

Enfin, nous allons parler de la routine pour afficher les recettes. Nous allons créer un fichier HTML très simple et l'ouvrir avec le navigateur par défaut pour permettre l'impression à partir de celui-ci. C'est pourquoi nous importons la bibliothèque webbrowser. Dans la routine du menu, option 6, insérez le code affiché en haut de la page suivante (Ndt : Code erroné dans la version anglaise et pas encore affiché sur le site de l'auteur).

À nouveau, nous affichons une liste de toutes les recettes et permettons à l'utilisateur de choisir celle qu'il souhaite imprimer. Nous appelons la routine ImprimeTout dans la classe LivreDeRecettes. Ce code est affiché en bas à droite de la page suivante.

Nous commençons par la commande : « fi = open([filename],'w') » qui crée le fichier, puis nous récupérons les informations de la table recette et les écrivons dans le fichier avec la commande fi.write. Nous utilisons l'étiquette en-tête 1 <H1></H1> pour le titre, l'étiquette <H2> pour le nombre de personnes et la source. Nous utilisons les étiquettes liste pour la liste des ingrédients, puis nous écrivons les instructions. À part cela, ce ne sont que de simples requêtes que nous avons déjà apprises. Enfin, nous fermons le fichier avec la commande fi.close() et utilisons webbrow-ser.open([filename]) pour le fichier que nous venons de créer. L'utilisateur peut alors imprimer à partir de son navigateur Web, si nécessaire.

Whoua ! C'était notre plus grosse application à ce jour. J'ai posté le code source complet (en anglais) (et l'échantillon de base de données si vous l'avez raté le mois dernier) sur mon site Web. Si vous ne voulez pas le retaper complètement ou si vous avez des problèmes, faites un saut sur mon site : www.thedesignatedgeek.com pour récupérer le code.

```
cbk.AfficheTouT()
print '0 - Retour au menu'
try:
res = int(raw_input('Choisissez une recette à
SUPPRIMER ou 0 pour quitter -> '))
if res != 0:
cbk.Supprime1Recette(res)
elif res == '0':
print 'Retour au menu...'
else:
print 'Commande inconnue. Retour au menu.'
except ValueError:
print "Ce n'est pas un nombre...retour au
menu."
```

```
def Supprime1Recette(self,laquelle):
resp = raw_input('Êtes-vous sûr de vouloir
supprimer cet enregistrement ? (O/n) -> ')
if string.upper(resp) == 'O':
sql = "DELETE FROM Recettes WHERE pkID = %s" %
str(laquelle)
curseur.execute(sql)
sql = "DELETE FROM Instructions WHERE
idRecette = %s" % str(laquelle)
curseur.execute(sql)
sql = "DELETE FROM Ingredients WHERE idRecette
= %s" % str(laquelle)
curseur.execute(sql)
print "Données de la recette SUPPRIMÉES"
resp = raw_input('Appuyez sur une touche -> ')
else:
print "Suppression annulée - Retour au menu"
```

```
cbk.Supprime1Recette()
print '0 - Retour au menu'
try:
res = int(raw_input('Choisissez une recette à SUPPRIMER ou 0 pour quitter -> '))
if res != 0:
cbk.Supprime1Recette(res)
elif res == '0':
print 'Retour au menu...'
else:
print 'Commande inconnue. Retour au menu.'
except ValueError:
print "Ce n'est pas un nombre...retour au menu."
```

```
def ImprimeTout(self,laquelle):
fi = open('afficheRecettes.html','w')
sql = "SELECT * FROM Recettes WHERE pkID = %s" % laquelle
for x in curseur.execute(sql):
nomRecette = x[1]
SourceRecette = x[3]
NbPersonnesRecette = x[2]
fi.write("<H1>%s</H1>" % NomRecette)
fi.write("<H2>Source: %s</H2>" % SourceRecette)
fi.write("<H2>Convives: %s</H2>" % NbPersonnesRecette)
fi.write("<H3> Liste d'ingrédients : </H3>")
sql = 'SELECT * FROM Ingredients WHERE idRecette = %s' % laquelle
for x in curseur.execute(sql):
fi.write("<li>%s</li>" % x[1])
fi.write("<H3>Instructions :</H3>")
sql = 'SELECT * FROM Instructions WHERE idRecette = %s' % laquelle
for x in curseur.execute(sql):
fi.write(x[1])
fi.close()
webbrowser.open('afficheRecettes.html')
print "Fin"
```

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume deux

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Full Circle Magazine spécial

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

Il ne s'agit de rien d'autre qu'une reprise de la série Programmer en Python, parties 9 à 16, numéros 35 à 42 ; pas de chichi, juste les faits.

Gardez à l'esprit la date de publication ; les versions actuelles du matériel et des logiciels peuvent être différentes de celles illustrées. Il vous est recommandé de bien vérifier la version de votre matériel et des logiciels avant d'essayer d'émuler les tutoriels dans ces numéros spéciaux. Il se peut que vous ayez des logiciels plus récents ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

Site web :
http://www.fullcirclemagazine.org/

Forums :
http://ubuntuforums.org/forumdisplay.php?f=270

IRC : #fullcirclemagazine on chat.freenode.net

Équipe éditoriale :

Rédacteur en chef : Ronnie Tucker (aka: RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia (aka: admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcaster : Robin Catling (aka RobinCatling)
podcast@fullcirclemagazine.org
Dir. Comm : Robert Clipsham (aka: mrmonday) -
mrmonday@fullcirclemagazine.org

TUTORIEL

Programmer en Python - Partie 9

Greg Walters

Si vous êtes comme moi, vous avez sur votre ordinateur quelques-uns de vos morceaux de musique favoris sous forme de fichiers MP3. Tant que vous avez moins de 1000 fichiers de musique, il est assez facile de vous souvenir de ce que vous avez et où ça se trouve. En revanche, moi, j'en ai beaucoup plus que ça. Dans une vie antérieure, j'étais DJ et j'ai converti la plupart de ma musique il y des années. Le plus gros problème que j'ai rencontré a été l'espace disque. Maintenant, le plus gros problème est d'arriver à me souvenir de ce que j'ai et où ça se trouve.

Dans cet article et le suivant, nous verrons comment fabriquer un catalogue de nos fichiers MP3. Nous apprendrons également de nouveaux concepts Python et nous reverrons nos connaissances en matière de bases de données.

Tout d'abord, un fichier MP3 peut contenir des informations sur le fichier lui-même. Le titre de la chanson, l'album, l'artiste, et bien plus encore. Ces informations sont stockées dans des balises ID3 et on les appelle des méta-données. Au tout début, on ne pouvait stocker qu'une quantité très limitée d'informations dans un fichier MP3. À l'origine, elle se trouvait à la fin du fichier dans un bloc de 128 octets. À cause de la petite taille de ce bloc, le titre de la chanson devait faire moins de 30 caractères, le nom de l'artiste également, et tout le reste aussi. Cela convenait pour beaucoup de fichiers de musique, mais (et c'est l'une de mes chansons préférées) quand vous aviez une chanson intitulée « Clowns (The Demise of the European Circus with No Thanks to Fellini) », vous ne pouviez conserver que les 30 premiers caractères. C'était extrêmement frustrant pour beaucoup de gens. Alors le standard ID3 pour les balises fut renommé ID3v1, et un nouveau format fut créé, appelé – devinez comment - ID3v2. Ce nouveau format permettait d'avoir des informations de longueur variable qui se trouvaient en début de fichier, tandis que les anciennes méta-données au format ID3v1 restaient en fin de fichier, permettant aux vieux lecteurs de fonctionner. Désormais, le conteneur de méta-données acceptait jusqu'à 256 Mo de données. C'était idéal pour les stations de radio et les fous comme moi. Avec ID3v2, chaque groupe d'informations est placé dans ce qu'on appelle un cadre et chaque cadre a un identifiant. Dans une version antérieure de ID3v2, l'identifiant avait 3 caractères. La version actuelle (ID3v2.4) utilise des identifiants de 4 caractères.

Dans les premiers temps, on ouvrait les fichiers en mode binaire et on fouillait le fichier pour trouver l'information qui nous intéressait, mais c'était un gros travail, parce qu'il n'y avait pas de bibliothèque standard pour s'occuper de ça. Maintenant, il existe un certain nombre de bibliothèques pour gérer cela à notre place. Pour notre projet, nous allons en utiliser une qui s'appelle Mutagen. Il vous faudra aller dans Synaptic et installer python-mutagen. Si vous voulez, vous pouvez rechercher « ID3 » dans Synaptic. Vous verrez qu'il y a plus de 90 paquets (dans Karmic), et si vous tapez « python » dans le champ de recherche rapide vous trouverez 8 paquets. Chacun d'eux a des avantages et des inconvénients, mais pour notre projet nous utiliserons Mutagen. Cela dit, vous pouvez toujours essayer les autres pour en apprendre davantage.

Maintenant que Mutagen installé, commençons à coder.

Démarrez un nouveau projet et appelez-le « mCat ». Nous allons commencer par les « import ».

```
from mutagen.mp3 import MP3

import os

from os.path import
join,getsize,exists

import sys

import apsw
```

Vous avez déjà rencontré la plupart de ces instructions dans les articles précédents. Maintenant, nous allons créer les en-têtes de nos fonctions.

```
def FabriquerDataBase():
      pass
def S2HMS(t):
      pass
def ParcourirChemin(chemin):
      pass
```

```
def error(message):
    pass
def main():
    pass
def usage():
    pass
```

Ah ! quelque chose de nouveau. Nous avons maintenant une fonction principale (« main ») et une fonction « usage ». À quoi servent-elles ? Introduisons encore une chose avant de parler de ça.

```
if __name__ == '__main__':
    main()
```

Qu'est-ce que c'est que ça ?

C'est un truc qui permet d'utiliser notre fichier soit comme une application autonome, soit comme un module réutilisable qui pourra être importé dans une autre application. Cela dit simplement : « Si ce fichier est l'application principale, il faut aller exécuter la routine main, sinon c'est que nous utiliserons ce programme comme un module utilitaire et que les fonctions seront appelées directement depuis un autre programme. »

Ensuite, occupons-nous de remplir la fonction « usage ». Le code est présenté ci-dessous.

Maintenant nous allons créer un message à afficher à l'intention de l'utilisateur s'il ne démarre pas l'application avec un paramètre dont nous avons besoin pour pouvoir exécuter le programme en tant qu'application autonome. Notez que nous utilisons « \n » pour forcer le passage à la ligne et « \t » pour forcer une tabulation. Nous utilisons également un « %s » pour inclure le nom de l'application qui se trouve dans sys.argv[0]. Nous utilisons ensuite la routine d'erreur pour afficher le message, puis sortir de l'application (sys.exit(1)).

Voyons maintenant le contenu de la routine d'erreur. Voici son code complet.

```
def error(message):
    print >> sys.stderr,
    str(message)
```

Nous utilisons ici ce qu'on appelle une redirection (le « "»" »). Quand on appelle la fonction « print », on dit à python que l'on veut afficher, ou envoyer un flux, sur la sortie standard, en général le terminal dans lequel on a lancé le programme. Pour cela, on utilise (de façon cachée) stdout. Lorsqu'on veut envoyer un message d'erreur, on utilise le flux stderr, qui est aussi par défaut le terminal. Donc on redirige la sortie de « print » vers le flux stderr.

Maintenant, penchons-nous sur la routine principale. Nous allons régler la connexion et le curseur pour notre base de données, puis regarder les paramètres passés en arguments et, si tout va bien, nous appellerons nos fonctions pour faire le vrai travail du programme. Voici le code (ci-dessous, en bas de page).

Comme la dernière fois, on crée deux variables globales appelées connexion et curseur pour notre base de données. Puis nous regardons les paramètres (s'il y en a) passés sur la ligne de commande dans le terminal. On utilise pour cela la commande sys.argv et on cherche ici deux paramètres : le nom de l'application qui est automaiquement réglé et le chemin vers nos fichiers MP3. Si on ne trouve pas ces deux paramètres, on saute dans la routine « usage » qui affiche notre message à l'écran et sort du programme. Si on les trouve, on passe par la clause « else » de

```
def usage():
    message = (
        '=============================================\n'
        'mCat - Cherche tous les fichiers *.mp3 dans un répertoire donné et ses sous-répertoires,\n'
        '\tlit les données id3, et écrit ces informations dans une base SQLite.\n\n'
        'Usage:\n'
        '\t{0} <nomRepertoire>\n'
        '\t où <nomRepertoire> est le chemin vers les fichiers MP3.\n\n'
        'Auteur: Greg Walters\n'
        'pour le Full Circle Magazine\n'
        '=============================================\n'
        ).format(sys.argv[0])
    error(message)
    sys.exit(1)
```

```
def main():
    global connexion
    global curseur
    #-------------------------------------------------
    if len(sys.argv) != 2:
        usage()
    else:
        RepertoireDepart = sys.argv[1]
        if not exists(RepertoireDepart): # From os.path
            print('Le chemin {0} n'existe pas...Fin
du programme.').format(RepertoireDepart)
            sys.exit(1)
        else:
            print('Prêt à traiter le répertoire {0}
:').format(RepertoireDepart)
        # on crée la connexion et le curseur
        connexion=apsw.Connection("mCat.db3")
        curseur=connexion.cursor()
        # on fabrique la base de données si elle
n'existe pas
        FabriquerBase()
        # on fait le boulot
        ParcourirChemin(RepertoireDepart)
        # on ferme le curseur et la connexion...
        curseur.close()
        connexion.close()
        # on annonce qu'on a terminé
        print("FIN !")
```

notre instruction « if ». Puis on place le paramètre représentant le chemin dans la variable RepertoireDebut.

Comprenez bien que si le chemin contient une espace, par exemple « /mnt/musique/Adulte Contemporain », les caractères suivant l'espace seront vus comme un autre paramètre. Il faut donc s'assurer de mettre des guillemets doubles lorsqu'on utilise un chemin avec des espaces. On règle ensuite la connexion et le curseur, on crée la base de données, puis on fait le travail principal dans la routine ParcourirChemin, et finalement on ferme le curseur et la connexion à la base, et on dit à l'utilisateur que c'est terminé. Le code complet de la routine ParcourirChemin est ici : http://pastebin.com/UeY3JYg7.

D'abord on efface les trois compteurs que nous utiliserons pour garder la trace du travail accompli. Puis nous ouvrons un fichier qui contiendra le journal d'erreurs en cas de problème. Ensuite nous faisons un parcours récursif du chemin fourni par l'utilisateur. Nous commençons simplement par le chemin fourni, puis entrons et sortons de chaque sous-répertoire qui se trouve là, en cherchant des fichiers dont l'extension est « .mp3 ». Ensuite nous incrémentons le compteur de répertoires puis le compteur de fichiers pour garder la trace du nombre de fichiers traités. Puis nous examinons chaque fichier. On efface les variables locales contenant l'information sur chaque chanson. On utilise la fonction « join » de os.path pour créer un chemin propre vers le fichier pour pouvoir dire à Mutagen où se trouve le fichier. Maintenant on passe le nom de fichier à la classe MP3 et on récupère une instance de « audio ». Puis on récupère toutes les étiquettes ID3 contenues dans le fichier et on parcourt cette liste pour récupérer les valeurs des étiquettes qui nous intéressent et les assigner à nos variables temporaires. De cette façon, on fait peu d'erreurs. Regardez le morceau de code traitant du numéro de piste. Quand Mutagen renvoie un numéro de piste, ça peut être une simple valeur, comme « 4/18 » ou comme _trk[0] et _trk[1], ou bien rien du tout. On utilise un wrapper « try/except » pour rattraper les erreurs éventuelles dues à cela. Regardez maintenant comment on écrit les enregistrements. On fait un peu différemment de la dernière fois. Ici, on crée la requête SQL comme avant, mais cette fois-ci on remplace les valeurs par « ? ». On place ensuite les valeurs dans l'instruction curseur.execute(). Selon le site web de APSW, c'est la meilleure façon de faire, alors je ne vais pas me battre avec eux. Et enfin on traite les autres types d'erreurs qui peuvent survenir. Pour la majeure partie, ce seront des TypeErrors ou des ValueErrors (erreurs de types ou de valeurs) et seront probablement dues à des caractères unicode qui ne sont pas gérés. Jetez un coup d'oeil rapide à la façon étrange que nous utilisons pour mettre en forme et afficher la chaîne de caractères. Nous n'utilisons pas le caractère de substitution « % », mais une substitution de type {0} qui fait partie de Python 3.x. La forme de base est la suivante :

```
Print('Chaîne à afficher
avec {0} nombre de
paramètres").format(valeurs
de remplacement)
```

Nous utilisons la syntaxe de base pour les lignes efile.writelines. Pour finir, nous devrions regarder la routine S2HMS. Elle prend en argument la longueur de la chanson sous forme d'un nombre réel à virgule flottante, tel que retourné par Mutagen, et le convertit en chaîne de caractères sous la forme « Heure:Minutes:Secondes » ou « Minutes:Secondes ». Regardez l'instruction « return ». Une fois encore, on utilise un formatage Python 3.x. Cependant il y a quelque chose de nouveau : on utilise trois ensembles de substitution (0, 1 et 2), mais quel est donc ce « :02n » après les nombres 1 et 2 ? Cela signifie que l'on veut des nombres sur deux chiffres avec des 0 au début. Ainsi, si une chanson dure 2 minutes et 4 secondes, la chaîne retournée sera « 2:04 » et non pas « 2:4 ». Le code complet de notre programme est ici : http://pastebin.com/xdtPvVqH.

Cherchez sur le web et vous verrez que Mutagen fait bien plus que s'occuper des MP3.

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille.

MON HISTOIRE RAPIDE

Mon studio est entièrement numérique avec quatre machines sous Windows XP branchées en réseau pair-à-pair. Ma cinquième machine tourne sous Linux Ubuntu 9.04 exclusivement, en tant que machine pour des tests sous Linux. J'ai commencé avec Ubuntu 7.04 et j'ai fait les mises à jour à chaque nouvelle version. Je lai trouvé et je le trouve encore très stable, facile d'utilisation et de configuration, car chaque version améliore le système.

Pour le moment, c'est seulement une machine où je fais des tests, mais elle est reliée à mon réseau et partage des données avec les machines sous Windows. Je suis vraiment content de la stabilité d'Ubuntu côté mises à jour, programmes, matériel pris en charge et mises à jour de pilotes. Il est vraiment malencontreux que les grands éditeurs comme Adobe ne fassent pas de portage, mais Wine semble bien fonctionner. Il y a des logiciels graphiques et des imprimantes professionnelles en lien avec mon équipement photographique qui ne fonctionnent pas et je devrai donc attendre que Wine s'améliore ou que les logiciels soient portés sous Linux.

L'audio, la vidéo, les CD/DVD, l'USB et les lecteurs Zip semblent tous fonctionner dès l'installation, ce qui est agréable. Il reste quelques défauts côté logiciels, mais ce ne sont que des problèmes mineurs.

Tout compte fait, Ubuntu est original visuellement et je me suis bien amusé avec. Je ne suis pas un « geek » et je n'utilise donc pas la ligne de commande, à moins d'être curieux en lisant un tutoriel et en voulant essayer ; l'interface utilisateur est plutôt complète pour les non informaticiens qui veulent s'en tenir au mode graphique.

Je télécharge Full Circle Magazine chaque mois et j'en ai fait profiter l'un de mes collègues pour lui montrer ce qui est disponible. Beaucoup de personnes ne connaissent pas encore ce système d'exploitation et sa grande facilité d'utilisation, mais au fur et à mesure que les mécontents de Microsoft se passent le mot, je m'attends à voir une croissance plus importante. La chose que j'adore vraiment avec ce système est la possibilité de fermer un programme qui ne répond plus. Le bouton de fermeture fonctionne bien sous Linux et élimine la frustration que l'on ressent en attendant que les fenêtres ne soient plus figées sous XP Pourquoi Windows ne fait pas quelque chose d'aussi simple que ça ? J'ai rarement besoin d'utiliser ce bouton sous Linux de toute façon, ce qui montre combien Linux est stable.

Brian G Hartnell - *Photographe*

TUTORIEL

Greg Walters

Programmer en Python - Partie 10

Vous avez probablement entendu parler du XML. Mais vous ne savez peut-être pas de quoi il s'agit. Notre leçon de ce mois-ci aura pour thème le XML. Notre but est :

• de vous familiariser avec ce qu'est XML.

• de vous montrer comment lire et écrire des fichiers XML dans vos propres applications.

• de vous préparer pour un projet XML beaucoup plus gros la prochaine fois.

Alors... parlons de XML. XML signifie EXtensible Markup Language (langage extensible à balises), un peu comme HTML. Il a été conçu pour permettre de stocker et de transporter des données efficacement par internet ou d'autres moyens de communication. XML est tout simplement un fichier texte formaté en utilisant vos propres balises et qui devrait normalement être autodocumenté. En tant que fichier texte, on peut le compresser pour que le transfert des données soit plus facile et plus rapide. Au contraire du HTML, le XML ne fait rien par lui-même. Il ne s'occupe pas de la façon dont vous voulez afficher les données. Comme je l'ai dit plus tôt, XML ne vous oblige pas à utiliser un ensemble de balises standards. Vous pouvez créer les vôtres.

Regardons un exemple de fichier XML générique :

```
<racine>
   <noeud1>Des données
ici</noeud1>
   <noeud2 attribute="quelque
chose">données Noeud 2</noeud2>
   <noeud3>
      <noeud3sousnoeud1>en-
core des données</noeud3sous-
noeud1>
   </noeud3>
</racine>
```

La première chose à remarquer est l'indentation. En réalité, l'indentation n'est là que pour nous, humains. Le fichier XML fonctionnerait de la même façon s'il essemblait à ça :

```
<racine><noeud1>Des données
ici</noeud1><noeud2 attri-
bute="quelque chose">données
Noeud
2</noeud2><noeud3><noeud3sou-
noeud1>encore des données
</noeud3sousnoeud1></noeud3></
racine>
```

Ensuite, les balises contenues entre les crochets « <> » doivent suivre certaines règles. D'abord, elles doivent être formées d'un seul mot. Ensuite, lorsqu'on a une balise ouvrante (par exemple <racine>), on doit avoir une balise fermante qui lui correspond. La balise fermante commence par un « / ». Les balises sont également sensibles à la casse : <noeud>, <Noeud>, <NOEUD> et <NoeuD> sont toutes des balises différentes et la balise fermante doit correspondre. Les noms de balises peuvent contenir des lettres, des nombres et d'autres caractères, mais ne doivent pas commencer par un nombre ou un signe de ponctuation. Vous devriez éviter « - », « . » et « : » dans le nom de vos balises, car certains logiciels pourraient les considérer comme des commandes ou des propriétés d'un objet. En outre, les deux-points sont réservés pour autre chose. Les balises s'appellent aussi des éléments.

Chaque fichier XML est simplement un arbre, démarrant par une racine d'où partent des branches. Chaque fichier XML DOIT comprendre un élément racine, qui est le parent de tout le reste du fichier. Regardez à nouveau notre exemple. Après la racine, il y a trois éléments fils : noeud1, noeud2 et noeud3. Ils sont tous fils de l'élément racine, mais noeud3 est aussi un parent de noeud3sousnoeud1.

Maintenant, regardons noeud2. Remarquez qu'en plus d'avoir des données normales à l'intérieur des crochets, il a également quelque chose qu'on appelle un attribut. De nos jours, de nombreux développeurs évitent les attributs, car les éléments sont aussi efficaces et ça donne moins de tracas, mais vous découvrirez que les attributs sont toujours utilisés. Nous les étudierons plus en détail dans un moment.

Regardons l'exemple suivant, qui est très utile. Ici, nous avons l'élément racine appelé « gens », qui contient deux éléments fils appelés « individu ». Chaque enfant « individu » a

six éléments enfants : prénom, nom, sexe, adresse, ville, État. Au premier coup d'œil, ce fichier XML peut vous faire penser à une base de données (rappelez-vous les dernières leçons), et vous auriez raison. En fait, certaines applications utilisent des fichiers XML comme des structures simples de bases de données. Maintenant, nous allons pouvoir écrire une application pour lire ce fichier XML sans trop de difficultés. Il suffit d'ouvrir le fichier, de lire chaque ligne et, en fonction de l'élément, de s'occuper des données pendant leur lecture et enfin de fermer le fichier quand on a terminé. Cependant, il y a de meilleures façons pour faire ça.

```
<gens>
    <individu>
        <prénom>Samantha</prénom>
        <nom>Pharoh</nom>
        <sexe>Female</sexe>
        <adresse>123 Main St.</adresse>
        <ville>Denver</ville>
        <état>Colorado</état>
    </individu>
    <individu>
        <prénom>Steve</prénom>
        <nom>Levon</nom>
        <sexe>Male</sexe>
        <adresse>332120 Arapahoe Blvd.</adresse>
        <ville>Denver</ville>
        <état>Colorado</état>
    </individu>
</gens>
```

Dans les exemples qui suivent, nous allons utiliser une bibliothèque appelée ElementTree. Vous pouvez la récupérer directement avec Synaptic en installant python-elementtree. Cependant, j'ai choisi d'aller sur le site web de ElementTree (http://effbot.org/downloads/#elementtree) et de télécharger le fichier source directement (elementtree-1.2.6-20050316.tar.gz). Une fois téléchargé, j'ai utilisé le gestionnaire d'archives pour le décompresser dans un répertoire temporaire. Je me suis déplacé dans ce répertoire et j'ai lancé la commande : « sudo python setup.py install », qui a placé les fichiers dans le répertoire commun de commandes de python, me permettant de les utiliser avec python 2.5 et python 2.6. Maintenant on peut commencer à travailler. Créez un répertoire pour y placer le code de ce mois-ci, copiez les données XML ci-dessus dans votre éditeur de texte préféré et sauvegardez-les dans ce répertoire, sous le nom : « xmlexemple1.xml ».

Maintenant voyons le code. La première chose à faire est de tester l'installation d'ElementTree. Voici le code :

```
import
elementtree.Element-
Tree as ET
arbre =
ET.parse('xml
exemple1.xml')

ET.dump(arbre)
```

En lançant le programme de test, on devrait obtenir quelque chose comme ce qui se trouve ci-contre à droite.

Tout ce que nous avons fait a été de permettre à ElementTree d'ouvrir le fichier, de l'analyser pour voir de quoi il est composé et de l'afficher tel quel. Rien de bien folichon.

Maintenant, remplacez le code par ce qui suit :

```
import elementtree.Element-
Tree as ET

arbre = ET.parse('xm-
lexemple1.xml')

individu = arbre.fin-
dall('.//individu')

for i in individu:
    for donnee in i:
        print "Élement : %s -
Donnée : %s" %(don-
nee.tag,donnee.text)
```

```
/usr/bin/python -u
 "/home/greg/Documents/articles/xm
l/reader1.py"

<gens>
    <individu>
        <prénom>Samantha</prénom>
        <nom>Pharoh</nom>
        <sexe>Female</sexe>
        <adresse>123 Main St.
</adresse>
        <ville>Denver</ville>
        <état>Colorado</état>
    </individu>
    <individu>
        <prénom>Steve</prénom>
        <nom>Levon</nom>
        <sexe>Male</sexe>
        <adresse>332120 Arapahoe
Blvd.</adresse>
        <ville>Denver</ville>
        <état>Colorado</état>
    </individu>
</gens>
```

et exécutez-le à nouveau. Maintenant vous devriez obtenir :

```
/usr/bin/python -u
 "/home/greg/Documents/articles/xml/reader1.py"

Élement : prénom - Donnée : Samantha
Élement : nom - Donnée : Pharoh
Élement : sexe - Donnée : Female
Élement : adresse - Donnée : 123 Main St.
Élement : ville - Donnée : Denver
Élement : état - Donnée : Colorado
Élement : prénom - Donnée : Steve
Élement : nom - Donnée : Levon
Élement : sexe - Donnée : Male
Élement : adresse - Donnée : 332120 Arapahoe Blvd.
Élement : ville - Donnée : Denver
Élement : état - Donnée : Colorado
```

Maintenant, on obtient chaque donnée avec le nom de la balise. On peut obtenir facilement un bel affichage avec ce qu'on a. Regardons ce que nous avons fait. ElementTree a analysé le fichier pour obtenir un arbre, puis on lui a demandé de trouver toutes les instances de « individu ». Dans l'exemple que nous utilisons, il y en a 2, mais il pourrait y en avoir 1 ou 1000. « Individu » est un fils de « Gens » et nous savons que « Gens » est la racine. Toutes nos données sont contenues dans « Individu ». Ensuite, nous avons créé une boucle « for » simple pour parcourir chaque objet « individu ». Puis une autre boucle « for » pour récupérer les données pour chaque individu et les afficher en montrant le nom de la balise (.tag) et les données (.text).

Maintenant, voyons un exemple plus réaliste. Ma famille et moi adorons une activité appelée « Geocaching ». Si vous ne savez pas de quoi il s'agit, c'est une sorte de chasse au trésor pour geeks qui utilise un GPS portatif pour trouver quelque chose que quelqu'un d'autre a caché. On récupère des coordonnées GPS brutes sur un site web, parfois avec des indices, et on entre les coordonnées dans son GPS pour essayer d'aller sur place trouver l'objet. D'après Wikipedia, il y a plus d'un million d'objets cachés de par le monde, il y en a donc sûrement quelques-uns près de chez vous. J'utilise deux sites web pour récupérer les localisations à découvrir : http://www.geocaching.com/ et http://navicache.com/. Il y en ad'autres, mais ces deux-là sont les plus fournis.

Les fichiers qui contiennent l'information sur chaque site de « geocaching » sont en général des fichiers XML basiques. Il existe des applications qui prennent ces données et les transfèrent dans le GPS. Certaines agissent comme des bases de données, ce qui vous permet de garder une trace de votre activité, parfois sur des cartes. Pour le moment, nous nous concentrerons sur une simple analyse des fichiers téléchargés.

Je suis allé sur Navicache et j'ai trouvé une cache récente au Texas. L'information contenue dans le fichier est sur la page précédente, à gauche.

Copiez les données de ce cadre et sauvegardez-les dans le fichier « Cache.loc ». Avant de commencer à coder, examinons ce fichier.

La première ligne nous dit simplement qu'il s'agit d'un fichier XML validé et nous pouvons l'ignorer. La

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<loc version="1.0" src="NaviCache">
   <waypoint>
      <name id="N02CAC"><![CDATA[Take Goofy Pictures at Grapevine Lake by g_phillips
Open Cache: Unrestricted
Cache Type: Normal
Cache Size: Normal
Difficulty: 1.5
Terrain   : 2.0]]></name>
      <coord lat="32.9890166666667" lon="-97.0728833333333" />
      <type>Geocache</type>
      <link text="Cache Details">http://www.navicache.com/cgi-
bin/db/displaycache2.pl?CacheID=11436</link>
   </waypoint>
</loc>
```

Dossier Navicache

ligne suivante (qui commence par « loc ») est notre racine, et contient les attributs « version » et « src ». Rappelez-vous que je vous ai dit que les attributs sont utilisés dans certains fichiers. Nous verrons d'autres attributs dans ce fichier en continuant. Encore une fois, la racine peut être ignorée dans cet exemple. La ligne suivante nous donne le fils « waypoint ». Il s'agit d'un point de passage, en l'occurrence la localisation où la cache se trouve. Et voici maintenant les données importantes que nous attendions. Il y a le nom de la cache, les coordonnées, latitude et longitude, le type de cache et un lien vers la page web qui contient plus de renseignements sur cette cache. L'élément « name » (nom) est une longue chaîne de caractères qui contient plein d'informations utilisables, mais nous devrons l'analyser nous-mêmes. Maintenant, créons une nouvelle application pour lire et afficher ce fichier. Nommez-la « lireunecache.py ». Commencez par l'importation et les instructions d'analyse de l'exemple précédent.

```
import elementtree.Element-
Tree as ET

arbre = ET.parse('Cache.loc')
```

Maintenant nous voulons récupérer seulement les données de la balise « waypoint ». Pour cela, nous utilisons la fonction « .find » de ElementTree. Le résultat sera retourné dans l'objet « w » :

```
w = arbre.find('.//waypoint')
```

Ensuite, nous voulons parcourir toutes les données. Pour cela, nous utilisons une boucle « for ». Dans cette boucle, nous vérifions les balises pour trouver les éléments « name », « coord », « type » et « link ». En fonction de chaque balise trouvée, nous récupérons l'information pour l'afficher plus tard.

```
for w1 in w:
if w1.tag == "name":
```

Puisque nous chercherons la balise « name » en premier, regardons les données que nous obtiendrons :

```
<name id="N02CAC"><![CDA-
TA[Take Goofy Pictures at Gra-
pevine Lake by g_phillips

Open Cache: Unrestricted

Cache Type: Normal

Cache Size: Normal

Difficulty: 1.5

Terrain : 2.0]]></name>
```

C'est une chaîne vraiment très longue. L'identifiant de la cache est réglé en attribut. Le nom est la partie située après CDATA et avant la partie « Open cache: ». Nous allons découper la chaîne pour obtenir les petites portions qui nous intéressent. Nous pouvons obtenir une partie d'une chaîne en utilisant :

```
nouvellechaine = ancienne-
chaine[début:fin]
```

Ainsi, nous pouvons utiliser le code ci-dessous pour récupérer l'information dont nous avons besoin.

Ensuite, nous devons récupérer l'identifiant situé dans l'attribut de la balise « name ». On vérifie qu'il y a bien des attributs (on sait qu'il y en a), comme ceci :

```
if w1.keys():
for nom,valeur in
w1.items():
if nom == 'id':
CacheID = valeur
```

Maintenant on peut s'occuper des autres balises pour les coordonnées, le type et le lien avec le code (page suivante, premier à gauche. (Finalement, on affiche tout ça en utilisant le code (page suivante, deuxième en bas à gauche). Le code complet se trouve sur la page suivante à droite.

Vous en avez maintenant appris assez pour lire la plupart des fichiers XML. Comme toujours, vous pouvez récupérer le code complet de cette leçon sur mon site web : http://www.thedesignatedgeek.com.

```
# nom de la cache : texte jusqu'à "Open Cache: "
NomCache = w1.text[:w1.text.find("Open Cache: ")-1]
# trouver le texte entre "Open Cache: " et "Cache Type: "
OpenCache = w1.text[w1.text.find("Open Cache:
")+12:w1.text.find("Cache Type: ")-1]
# et ainsi de suite
TypeCache = w1.text[w1.text.find("Cache Type:
")+12:w1.text.find("Cache Size: ")-1]
TailleCache = w1.text[w1.text.find("Cache Size:
")+12:w1.text.find("Difficulty: ")-1]
Difficulte= w1.text[w1.text.find("Difficulty:
")+12:w1.text.find("Terrain
: ")-1]
Terrain =
w1.text[w1.text.find("Terrain                    : ")+12:]
```

TUTORIEL - PROGRAMMER EN PYTHON - PARTIE 10

La prochaine fois, nous utiliserons nos connaissances en XML pour récupérer de l'information à partir d'un merveilleux site de météo et l'afficher dans un terminal.

Amusez-vous bien !

```
elif w1.tag == "coord":
   if w1.keys():
      for nom,valeur in w1.items():
         if nom == "lat":
            Lat = valeur
         elif nom == "lon":
            Lon = valeur
elif w1.tag == "type":
   GType = w1.text
elif w1.tag == "link":
   if w1.keys():
      for nom,valeur in w1.items():
         Info = valeur
   Link = w1.text
```

```
print "Nom Cache : ",NomCache
print "ID Cache : ",IDCache
print "Open Cache : ",OpenCache
print "Type Cache : ",TypeCache
print "Taille Cache : ",TailleCache
print "Difficulté : ", Difficulte
print "Terrain : ",Terrain
print "Lat : ",Lat
print "Lon : ",Lon
print "GType : ",GType
print "Link : ",Link
```

```
import elementtree.ElementTree as ET
arbre = ET.parse('Cache.loc')
w = arbre.find('.//waypoint')
for w1 in w:
    if w1.tag == "name":
        # récupérer le nom : texte jusqu'à "Open Cache: "
        NomCache = w1.text[:w1.text.find("Open Cache: ")-1]
        # récupérer le texte entre "Open Cache: " et "Cache
Type: "
        OpenCache = w1.text[w1.text.find("Open Cache:
")+12:w1.text.find("Cache Type: ")-1]
        # et ainsi de suite
        TypeCache = w1.text[w1.text.find("Cache Type:
")+12:w1.text.find("Cache Size: ")-1]
        TailleCache = w1.text[w1.text.find("Cache Size:
")+12:w1.text.find("Difficulty: ")-1]
        Difficulte= w1.text[w1.text.find("Difficulty:
")+12:w1.text.find("Terrain   : ")-1]
        Terrain =    w1.text[w1.text.find("Terrain: ")+12:]
        if w1.keys():
            for nom,valeur in w1.items():
                if nom == 'id':
                    IDCache = valeur
    elif w1.tag == "coord":
        if w1.keys():
            for nom,valeur in w1.items():
                if nom == "lat":
                    Lat = valeur
                elif nom == "lon":
                    Lon = valeur
    elif w1.tag == "type":
        GType = w1.text
    elif w1.tag == "link":
        if w1.keys():
            for nom,valeur in w1.items():
                Info = valeur
        Link = w1.text
print "Nom Cache : ",NomCache
print "ID Cache : ",IDCache
print "Open Cache : ",OpenCache
print "Type Cache : ",TypeCache
print "Taille Cache : ",TailleCache
print "Difficulté : ", Difficulte
print "Terrain : ",Terrain
print "Lat : ",Lat
print "Lon : ",Lon
print "GType : ",GType
print "Link : ",Link
print "="*25

print "terminé"
```

TUTORIEL

Programmer en Python - Partie 11

Greg Walters

La dernière fois, je vous avais promis que nous utiliserions nos compétences en XML pour récupérer de l'information météo à partir d'un site web pour l'afficher dans un terminal. Nous y voici aujourd'hui.

Nous utiliserons une API de www.wunderground.com. J'entends la question : « Qu'est-ce qu'une API ? » monter dans votre gorge. API signifie Application Programming Interface (Interface de Programmation Applicative). Ce n'est qu'une phrase compliquée pour signifier que c'est une façon de s'interfacer avec une autre application. Pensez aux bibliothèques que nous importons. Certaines d'entre elles peuvent être exécutées en tant qu'applications autonomes, mais si on les importe en tant que bibliothèques, on peut utiliser la plupart de leurs fonctions dans nos programmes et ainsi on peut utiliser le code de quelqu'un d'autre. Dans le cas présent, nous utiliserons des adresses URL spécialement formatées pour interroger le site wunderground au sujet d'informations météorologiques, mais sans utiliser de navigateur. Certains diraient qu'une API est comme une petite porte cachée dans un autre programme, que le(s) programmeur(s) mettent là exprès pour notre usage. En tout cas, c'est l'extension d'une application pour qu'on puisse l'utiliser dans d'autres applications.

Cela semble curieux ? Eh bien, lis la suite, mon cher padawan.

Ouvrez votre navigateur favori et rendez-vous sur www.wunderground.com. Maintenant, faites une recherche de votre code postal ou votre ville ou État ou pays. On trouve une surabondance d'informations. Maintenant, allons sur la page de l'API : http://wiki.wunderground.com/index.php/API_-_XML.

Une des premières choses que vous remarquerez est qu'il y a des conditions d'utilisation de l'API. Veuillez les lire attentivement. Elles ne sont pas ardues et sont très simples à respecter. Nous allons nous intéresser aux fonctionnalités GeoLookupXML, WXCurrentObXML, AlertsXML et ForecastXML. Prenez le temps de les parcourir.

Passons sur la routine GeoLookupXML. Regardez ça tout seul. Nous nous concentrerons sur deux autres commandes : WXCurrent ObXML (les conditions actuelles) cette fois-ci, et ForecastXML (les prévisions) la prochaine fois.

Voici le lien pour WXCurrentObXML : http://api.wunderground.com/auto/wui/geo/WXCurrentObXML/index.xml?query=80013

Remplacez le zip-code américain 80013 par votre propre code postal ou, si vous êtes en dehors des États-Unis, vous pouvez essayer une ville et un pays, par exemple Paris, France, ou Londres, Angleterre.

Et voici le lien pour ForecastXML : http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=80013

À nouveau, remplacez le zip-code américain 80013 par votre propre code postal ou ville et pays. Essayons avec ces informations. Collez l'adresse dans votre navigateur favori. Vous recevrez en retour un grand nombre d'informations. Je vous laisse décider ce qui est vraiment important pour vous, mais nous allons regarder quelques-uns de ces éléments. Pour notre exemple, nous regarderons les balises suivantes :

display_location (affichage localisation)
observation_time (heure observation)
weather (météo)
temperature_string (température)
relative_humidity (humidité relative)
wind_string (vent)
pressure_string (pression atmosphérique)

Vous pouvez, bien entendu, ajouter d'autres balises qui vous intéressent. Cependant, ces quelques balises suffiront pour cet exemple et vous permettront d'aller plus loin par la suite.

Maintenant que nous savons ce que nous devons rechercher, commençons à coder notre application. Regardons les grandes lignes du programme.

Tout d'abord, nous vérifions ce

que l'utilisateur nous a demandé de faire. Si une localisation est passée en argument on va l'utiliser, sinon nous utiliserons la localisation par défaut que nous codons dans la routine principale. Nous passons ensuite cela à la routine « get Currents ». On utilise la localisation pour construire la chaîne de requête à envoyer au site web. On utilise « urllib.urlopen » pour récupérer la réponse depuis internet, et on la place dans un objet, puis on passe cet objet à la fonction « parse » de la bibliothèque « ElementTree ». On ferme ensuite la connexion à internet et on commence à parcourir les balises. Quand on trouve une balise qui nous intéresse, on sauvegarde le texte dans une variable que l'on utilisera plus tard pour l'affichage. Une fois qu'on a toutes les données, on les affiche. Le concept est plutôt simple.

Commencez par nommer votre fichier w_currents.py. Voici la partie de code avec les « import » :

```
from xml.etree import ElementTree as ET

import urllib

import sys

import getopt
```

Ensuite, nous plaçons quelques lignes d'aide (en haut à droite) au dessus des « imports ». Vérifiez bien que vous utilisez les triples guillemets. Cela permet d'écrire un commentaire sur plusieurs lignes. Nous reviendrons là-dessus dans un moment.

Maintenant on crée l'ébauche de nos classes, ci-dessous à droite, et les routines principales que l'on voit sur la page suivante.

Vous vous souvenez de la ligne « if name » que nous avons vue dans les articles précédents. Si on utilise notre code en tant qu'application autonome, on lance la routine principale ; sinon on peut utiliser ce code en tant que partie d'une bibliothèque. Une fois dans la routine principale, on vérifie ce qu'on a reçu en arguments, s'il y en a.

Si l'utilisateur utilise le paramètre « -h » ou « –help », on affiche les lignes d'aide (commentées avec les triples guillemets) situées en bas du programme. Cela se fait avec la routine « usage » qui indique à l'application d'afficher « doc ».

Si l'utilisateur utilise le paramètre « -l » (localisation) ou « -z » (zipcode ou code postal), cela écrasera la

```
""" w_currents.py
Renvoie les conditions actuelles, meteo et alertes pour
un zipcode de WeatherUnderground.com.
Usage : python wonderground.py [options]
Options :
-h, --help Montre cette aide
-l, --localisation Ville ou Etat a utiliser
-z, --zip Zipcode a utiliser comme localisation

Exemples :
w_currents.py -h (montre ce message d'aide)
w_currents.py -z 80013 (utilise le zipcode 80013 comme
localisation)
"""
```

```
class CurrentInfo:
"""
Cette routine recupere les conditions actuelles au format
XML sur WeatherUnderground.com
en se basant sur le zipcode ou le code d'aeroport...
actuellement teste uniquement avec un zipcode ou un code
d'aeroport
Pour la localisation :
pour un zipcode, utiliser 80013 (sans guillemets)
pour un aeroport, utiliser "KDEN" (guillemets doubles)
pour une ville ou un etat (Etats-Unis), utiliser
"Aurora,%20CO" ou "Aurora,CO" (guillemets doubles)
pour une ville ou un pays, utiliser "London,%20England"
(guillemets doubles)
"""
def getCurrents(self,debuglevel,Localisation):
pass

def output(self):
pass
def DoIt(self,Location):
pass

#=======================================
# FIN DE LA CLASSE CurrentInfo()
#=======================================
```

localisation par défaut réglée en interne. Quand vous passez une localisation, vérifiez que vous utilisez les guillemets pour entourer la chaîne et n'utilisez pas d'espaces. Par exemple, pour récupérer les conditions actuelles à Dallas, Texas, utilisez « -l "Dallas,Texas" ».

Les lecteurs astucieux auront réalisé que le traitement de -z et de -l sont quasiment les mêmes. Vos pouvez modifier le -l pour vérifier qu'il n'y a pas d'espace et reformater la chaîne avant de l'envoyer à la routine. Vous devez savoir le faire maintenant.

Enfin, on crée une instance de notre classe « CurrentInfo » que nous appelons « currents », puis on envoie la localisation à la routine « DoIt ». Complétons-la maintenant :

```
def DoIt(self,Localisation):
self.getCurrents(1,Localisation)

self.output()
```

Très simple. On envoie la localisation et le niveau de débogage souhaité à la routine « getCurrents», puis on appelle la routine d'affichage. On aurait pu faire l'affichage directement dans la routine « getCurrents », mais de cette façon on améliore la flexibilité car on pourra afficher les informations de différentes façons si nécessaire.

Vous pouvez voir le code de la routine « getCurrents » à la page suivante.

Nous avons ici un paramètre appelé « debuglevel ». Ainsi, on peut afficher des informations utiles au cas où les choses ne se passent pas de la façon que nous souhaitons. Il sert également pendant les premières phases de codage. Une fois que vous aurez obtenu un programme qui fonctionne, vous pourrez retirer tout ce qui concerne « debuglevel ». Si vous allez diffuser votre code largement, ou si vous avez fait ce programme pour quelqu'un d'autre, assurez-vous de retirer ces parties de code et de tester à nouveau votre programme.

```
def usage():
print __doc__
def main(argv):
localisation = 80013
try:
opts, args = getopt.getopt(argv, "hz:l:", ["help=",
"zip=", "localisation="])
except getopt.GetoptError:
usage()
sys.exit(2)
for opt, arg in opts:
if opt in ("-h", "--help"):
usage()
sys.exit()
elif opt in ("-l", "--localisation"):
localisation = arg
elif opt in ("-z", "--zip"):
localisation = arg
print "Localisation = %s" % localisation
currents = CurrentInfo()
currents.DoIt(localisation)

#=========================================
# Boucle principale
#=========================================
if __name__ == "__main__":
main(sys.argv[1:])
```

```
<display_location>
<full>Aurora, CO</full>
<city>Aurora</city>
<state>CO</state>
<state_name>Colorado</state_name>
<country>US</country>
<country_iso3166>US</country_iso3166>
<zip>80013</zip>
<latitude>39.65906525</latitude>
<longitude>-104.78105927</longitude>
<elevation>1706.00000000 ft</elevation>
</display_location>
```

Maintenant, parlons du « try/except » que nous utilisons pour nous assurer que l'application ne plantera pas si quelque chose se passe mal. Dans la partie « try », on règle l'URL, ainsi qu'une limite de 8 secondes (urllib.socket.setdefaulttime-out(8)). On fait cela car, parfois, wunderground est occupé et ne répond pas. Ainsi, on ne reste pas planté là à attendre la connexion. Si vous souhaitez obtenir plus d'informations sur « urllib »,

```
def output(self):
print 'Information meteo depuis Wunderground.com'
print 'Info meteo pour %s ' % self.localisation
print self.heureobs
print 'Meteo actuelle - %s' % self.met
print 'Temp. actuelle - %s' % self.tmpB
print 'Pression atmospherique - %s' % self.baroB
print 'Humidite relative - %s' % self.humrel
print 'Vents %s' % self.vents
```

vous pouvez commencer par ceci : http://docs.python.org/library/urllib.html.

Si quelque chose d'inattendu se produit, on retombe dans la section « except » et on affiche un message d'erreur, puis on sort du programme (sys.exit(2)).

En supposant que tout fonctionne, on commence à rechercher nos balises. La première chose à faire est de trouver la localisation avec tree.findall("//full"). Souvenez-vous, « tree » est l'objet retourné par « ElementTree ». Voyez ci-dessous ce qui est renvoyé par l'API du site web.

C'est la première instance de la balise <full>, dans notre cas il s'agit de « Aurora, CO ». C'est ça que nous voulons utiliser comme localisation. Ensuite, on cherche « observation_time ». C'est l'heure à laquelle les conditions actuelles ont été enregistrées. On continue en cherchant toutes les données qui nous intéressent, en utilisant la même méthode.

En dernier lieu, occupons-nous de la routine d'affichage, que vous voyez à la page suivante.

Ici, on affiche simplement les variables.

Et c'est terminé. Vous pouvez voir un exemple d'affichage avec mon zipcode et le « debuglevel » réglé à 1, en bas à gauche de la page suivante.

Notez que j'ai choisi d'utiliser les balises qui contiennent à la fois les degrés Fahrenheit et Celsius. Si vous voulez, par exemple, n'afficher que les degrés Celsius, vous pouvez utiliser la balise <temp_c> à la place de <temperature _string>.

Le code complet peut être téléchargé ici : http://pastebin.com/jiyYnsWe.

```
def getCurrents(self,debuglevel,Localisation):
if debuglevel > 0:
print "Localisation = %s" % Localisation
try:
CurrentConditions =
'http://api.wunderground.com/auto/wui/geo/WXCurrentObXML
/index.xml?query=%s' % Localisation
urllib.socket.setdefaulttimeout(8)
usock = urllib.urlopen(CurrentConditions)
tree = ET.parse(usock)
usock.close()
except:
print 'ERREUR - Conditions actuelles - Ne peut
recuperer les informations sur le serveur...'
if debuglevel > 0:
print Localisation
sys.exit(2)
# affichage de la Localisation
for loc in tree.findall("//full"):
self.localisation = loc.text
# heure d'observation
for heure in tree.findall("//observation_time"):
self.heureobs = heure.text
# conditions actuelles
for meteo in tree.findall("//weather"):
self.met = meteo.text
# temperature
for TempF in tree.findall("//temperature_string"):
self.tmpB = TempF.text
# humidite
for hum in tree.findall("//relative_humidity"):
self.humrel = hum.text
# informations sur le vent
for vent in tree.findall("//wind_string"):
self.vents = vent.text
# pression atmospherique
for pression in tree.findall("//pressure_string"):
self.baroB = pression.text
```

getCurrents routine

Le mois prochain, nous nous concentrerons sur la partie prévisions de l'API.

D'ici là, amusez-vous bien !

```
Localisation = 80013
Information meteo depuis Wunderground.com
Info meteo pour Aurora, Colorado
Last Updated on June 16, 2:55 AM MDT
Meteo actuelle - Partly Cloudy
Temp. actuelle - 59 F (15 C)
Pression atmospherique - 29.81 in (1009 mb)
Humidite relative - 82%
Vents From the ESE at 9 MPH
Script terminated.
```

EXTRA ! EXTRA ! LISEZ CECI

LE SERVEUR PARFAIT ÉDITION SPECIALE

Il s'agit d'une édition spéciale du Full Circle qui est une réédition directe des articles Le Serveur parfait qui ont déjà été publiés dans le FCM n° 31 à 34.

http://fullcirclemagazine.org/special-edition-1-the-perfect-server/

Des éditions spéciales du magazine Full Circle sont sorties dans un monde sans méfiance*

PYTHON ÉDITION SPECIALE n° 1

Il s'agit d'une reprise de Programmer en Python, parties 1 à 8 par Greg Walters.

http://fullcirclemagazine.org/python-special-edition-1/

* Ni Full Circle magazine, ni ses concepteurs ne s'excusent pour l'hystérie éventuellement causée par la sortie de ces publications.

TUTORIEL

Greg Walters

Programmer en Python - Partie 12

Le mois dernier, nous avons utilisé l'API de wunderground et écrit du code pour récupérer les conditions météo actuelles. Cette fois-ci, nous allons nous occuper de la partie de l'API qui concerne les prévisions. Si vous n'avez pas pu lire les deux précédents articles sur l'installation de XML, et plus spécialement le dernier, vous devriez peut-être aller les parcourir avant de continuer. De la même façon qu'il y avait une adresse web pour récupérer les conditions actuelles, il y en a une pour les prévisions. Voici le lien vers la page XML des prévisions : http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?-query=80013

Comme précédemment, vous pouvez remplacer le « 80013 » par votre Ville/Pays, Ville/État ou code postal. Vous obtiendrez probablement environ 600 lignes de code XML. L'élément racine s'appelle « forecast » [Ndt : prévisions], et vous verrez quatre sous-éléments : « termsofservice », « txt_forecast », « simpleforecast » et « moon_phase ». Nous nous concentrerons sur « txt_forecast » et « simpleforecast ». Puisque nous avons déjà vu les sections « usage », « main » et « if name » la dernière fois, je vous laisse vous en occuper et je vais me concentrer sur ce dont nous aurons besoin aujourd'hui. Puisque je vous ai montré un extrait de « txt_forecast », commençons par là. En voici, ci-dessous, un tout petit morceau pour ma région.

Après l'élément parent « txt_forecast », nous récupérons la date, un élément « number », puis un élément appelé « forecastday » qui a ses propres fils : « period », « icon », « icons », « title » et quelque chose appelé « fct-text »... puis se répète. La première chose à remarquer est que, sous txt_forecast, la date n'est pas une date, mais une valeur de temps. Il s'avère que c'est le moment où la prévision a été publiée. La balise « number » indique combien de prévisions il y a pour les prochaines 24 heures. Je ne me souviens pas avoir déjà vu cette valeur en dessous de 2. Pour chaque prévision par période de 24 heures (<forecastday>), vous trouvez un numéro de période, une liste d'icônes, un titre (« Today », « Tonight », « Tomorrow » pour « aujourd'hui », « cette nuit », « demain ») ainsi que le texte d'une prévision simple. C'est un aperçu rapide des prévisions, en général pour les 12 prochaines heures.

Avant de commencer à travailler sur notre code, regardons la portion <simpleforecast> du fichier XML située à droite. Il y a une balise <forecastday> pour chaque jour de la période de prévision, en général 6 jours, aujourd'hui compris. Vous trouvez la date sous différents formats (j'aime personnellement la balise <pretty>), les températures maximales et minimales prévues en degrés Farenheit et Celsius, une prévision brute des conditions, diverses icônes, une icône pour le ciel (les conditions nuageuses à la station météo) et « pop » qui signifie « Probability Of Precipitation » [Ndt : probabilité de précipitation]. La balise <moon_phase> fournit des informations intéressantes sur le lever et le coucher du soleil et la lune. Maintenant, abordons le code. Voici les « import » :

```
from xml.etree import
Element-
```

```
<txt_forecast>
              <date>3:31 PM MDT</date>
              <number>2</number>
              -<forecastday>
                            <period>1</period>
                            <icon>nt_cloudy</icon>
                            +<icons></icons>
                            <title>Tonight</title>
                            -<fcttext>
                            Mostly cloudy with a 20
percent chance of thunderstorms in the evening...then
partly cloudy after midnight. Lows in the mid 40s.
Southeast winds 10 to 15 mph shifting to the south after
midnight.
                      </fcttext>
               </forecastday>
               +<forecastday></forecastday>
</txt_forecast>
```

```
Tree as ET
import urllib
import sys
import getopt
```

Maintenant, nous devons commencer à écrire notre classe. On crée une routine init pour déclarer et initialiser les variables dont nous aurons besoin ; regardez en haut à droite de la page suivante. Si vous ne voulez pas gérer à la fois les degrés Farenheit et Celsius, omettez la variable dont vous n'avez pas besoin. J'ai décidé de garder les deux.

Ensuite, nous commençons notre routine principale de récupération des données des prévisions. Regardez en bas à droite de la page suivante. Elle ressemble beaucoup à la routine des conditions actuelles sur laquelle nous avons travaillé la dernière fois. La seule différence majeure (pour l'instant) est l'URL que nous utilisons. Maintenant, les choses changent. Comme nous avons plusieurs fils dont la balise porte le même nom sous l'élément parent, nous devons modifier un peu les appels qui analysent ces balises. Le code se trouve en haut à gauche de la page suivante. Remarquez que nous utilisons « tree.find » cette fois-ci, et aussi des boucles « for » pour parcourir les données. C'est dommage que Python ne fournisse pas une commande SELECT/CASE comme d'autres langages, mais la routine IF/ELIF fonctionne bien, malgré qu'elle soit un peu plus maladroite. Maintenant, décomposons le code : on assigne tour à tour à la variable « previs » tout ce que contient la balise <txt_forecast> afin de récupérer tout le groupe de données ; puis on regarde les balises <date> et <number>, qui sont de premier niveau, et on charge les données dans nos variables. Maintenant les choses se compliquent un peu. Regardez à nouveau l'exemple de fichier XML retourné : il y a deux instances de <forecastday>, sous lesquelles les sous-éléments sont <period>, <icon>, <icons>, <title> et <fct-text>. On va boucler sur ces éléments et utiliser à nouveau l'instruction IF pour charger les valeurs dans nos variables.

Ensuite, nous allons regarder les données concernant les prévisions étendues pour les X prochains jours. Nous utilisons simplement la même méthode pour remplir nos variables ; regardez ci-dessus à droite. Maintenant il faut créer la routine d'affichage. Comme la dernière fois, elle sera plutôt générique. Vous en trouverez le code sur gauche. la page suivante, à Encore une fois, si vous ne voulez pas vous occuper des informations en degrés Celsius et Farenheit, modi-

```
<simpleforecast>
    -<forecastday>
        <period>1</period>
        -<date>
            <epoch>1275706825</epoch>
            <pretty_short>9:00 PM MDT</pretty_short>
            <pretty>9:00 PM MDT on June 04, 2010</pretty>
            <day>4</day>
            <month>6</month>
            <year>2010</year>
            <yday>154</yday>
            <hour>21</hour>
            <min>00</min>
            <sec>25</sec>
            <isdst>1</isdst>
            <monthname>June</monthname>
            <weekday_short/>
            <weekday>Friday</weekday>
            <ampm>PM</ampm>
            <tz_short>MDT</tz_short>
            <tz_long>America/Denver</tz_long>
        </date>
        -<high>
            <fahrenheit>92</fahrenheit>
            <celsius>33</celsius>
        </high>
        -<low>
            <fahrenheit>58</fahrenheit>
            <celsius>14</celsius>
        </low>
        <conditions>Partly Cloudy</conditions>
        <icon>partlycloudy</icon>
        +<icons>
        <skyicon>partlycloudy</skyicon>
        <pop>10</pop>
    </forecastday>
...
</simpleforecast>
```

```
#=======================================
# recupere les previsions pour aujourd'hui et (si
disponible) cette nuit
#=======================================
previs = tree.find('.//txt_forecast')
for f in previs:
    if f.tag == 'number':
        self.periodes = f.text
    elif f.tag == 'date':
        self.date = f.text
    for subelement in f:
        if subelement.tag == 'period':
            self.periode=int(subelement.text)
        if subelement.tag == 'fcttext':
            self.textePrevisions.append(subelement.tex
t)
        elif subelement.tag == 'icon':
            self.icone.append( subelement.text)
        elif subelement.tag == 'title':
            self.Titre.append(subelement.text)
```

```
class InfosPrevisions:
        def __init__(self):
                self.textePrevisions = []  # informations sur les
previsions
                self.Titre = []      # pour aujourd'hui et la
nuit prochaine
                self.date = ''
                self.icone = []      # icone a utiliser pour les
conditions meteo
                self.periodes = 0
                self.periode = 0
                #==============================
                # informations sur les previsions etendues
                #==============================
                self.extIcone = []    # icone a utiliser pour les

previsions etendues
                self.extJour = []     # nom du jour ("Monday",
"Tuesday" etc)
                self.extMaxi = []     # Temp. maxi (F)
                self.extMaxiC = []    # Temp. maxi (C)
                self.extMini = []     # Temp. mini (F)
                self.extMiniC = []    # Temp. mini (C)

                self.extConditions = []  # Conditions (texte)
                self.extPeriode = []   # information sur la
periode (compteur)
               self.extPrecip = []     # risque de precipitation
(pourcentage)
```

```
def GetDonneesPrevisions(self,localisation):
    try:
        donneesPrevisions = 'http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=%s' % localisation
        urllib.socket.setdefaulttimeout(8)
        usock = urllib.urlopen(donneesPrevisions)
        tree = ET.parse(usock)
        usock.close()
   except:
        print 'ERREUR - Previsions - Ne peut recuperer les informations sur le serveur...'
        sys.exit(2)
```

fiez le code pour afficher ce que vous souhaitez. Pour finir, voici la routine « DoIt » : def DoIt(self,Localisation,US,InclureAujd,Afficher):

```
self.GetDonneesPrevisions(Lo
-
calisation)
self.output(US,InclureAu-
jd,Afficher)
Maintenant nous pouvons
appeler la routine de cette
façon :
previsions =
InfosPrevisions()
previsions.DoIt('80013',1,0,
0)
# Insérez votre propre code
postal
```

C'est tout pour cette fois-ci. Je vous laisse gérer les alertes si vous souhaitez vous en occuper. Voici le code complet : http://pastebin.com/h5gzvyUr

Amusez-vous bien jusqu'à la prochaine fois.

Podcast Full Circle

AUDIO MP3 AUDIO OGG

Le ***Podcast Full Circle*** est de retour et meilleur que jamais !

Les thèmes de cet épisode sont :
• Actus - développement de Maverick
• Entrevue Lubuntu
• Jeux - Ed critique Osmos
• Retours
... et toute la bonne humeur habituelle.

Vos animateurs :
Robin Catling
Ed Hewitt
Ronnie Tucker

Le podcast et les notes sur l'émission sont visibles ici : http://fullcirclemagazine.org/

TUTORIEL Programmer en Python - Partie 13

Greg Walters

Ce mois-ci nous allons parler de l'utilisation de Curses avec Python. Non, il ne s'agit pas d'expliquer comment utiliser Python pour dire des gros mots [Ndt : « curses » signifie « grossièretés » en anglais], même si vous pouvez vous en servir ainsi si vous en avez vraiment besoin. Nous allons parler de l'utilisation de la bibliothèque Curses pour faire de jolis affichages à l'écran.

Si vous êtes suffisamment âgé pour vous souvenir des débuts de l'informatique, vous vous souviendrez qu'en entreprise, les ordinateurs étaient tous des ordinateurs centraux, avec de simples terminaux (écran et clavier) pour les entrées et les sorties. Vous pouviez avoir de nombreux terminaux connectés à un seul ordinateur. Le problème était que ces terminaux étaient des périphériques vraiment simplistes. Il n'y avait ni fenêtres, ni couleurs, ou quoi que ce soit (seulement 24 lignes de 80 caractères, au mieux). Quand les ordinateurs personnels sont devenus populaires, au bon vieux temps de DOS et CPM, c'est ce que vous aviez aussi. Quand les programmeurs ont travaillé pour avoir des écrans plus agréables (à cette époque), surtout pour la saisie de données et l'affichage, ils ont utilisé du papier à carreaux pour représenter l'écran. Chaque carré sur le papier représentait la position d'un caractère. Lorsque nous exécutons un programme Python dans un terminal, nous avons toujours un écran de taille 24×80. Cependant, cette limitation peut être facilement contournée en préparant bien les choses à l'avance. Alors, allez vite acheter quelques blocs de papier à carreaux dans un magasin près de chez vous.

Qu'importe, passons à la pratique, et créons notre premier programme avec Curses, visible ci-dessus à droite. Je donnerai les explications après que vous ayez jeté un coup d'œil au code.

```
#!/usr/bin/env python
# ExempleCurses1
#------------------------------
# Exemple Curses n°1
#------------------------------
import curses
monEcran = curses.initscr()
monEcran.border(0)
monEcran.addstr(12, 25, "Voyez comment Curses tourne !")
monEcran.refresh()
monEcran.getch()
curses.endwin()
```

Court mais simple. Examinons-le ligne par ligne. D'abord, on fait les « import », avec lesquels vous êtes maintenant familiers. Ensuite, on crée un nouvel objet « écran Curses », on l'initialise et on l'appelle monEcran (monEcran = curses.initscr()). Ceci est notre canevas, dans lequel nous allons peindre. Puis on utilise la commande monEcran.border(0) pour dessiner une bordure autour du canevas. Ce n'est pas obligatoire, mais l'écran sera plus joli. On utilise ensuite la méthode addstr pour écrire du texte sur le canevas, en commençant à la ligne 12 et à la position 25. Vous pouvez voir la méthode .addstr comme une instruction d'affichage de Curses. Enfin, la méthode refresh() rend notre travail visible. Si on ne rafraîchit pas l'écran, nos modifications ne seront pas visibles. Ensuite on attend que l'utilisateur appuie sur une touche (.getch), puis on libère l'objet « écran » (.endwin) pour permettre au terminal de reprendre la main. La commande curses.endwin() est TRÈS importante, car si on ne l'appelle pas, le terminal sera laissé dans un état vraiment bordélique. Alors assurez-vous d'appeler cette méthode avant la fin de votre programme.

Enregistrez ce programme sous le nom « Exemple-Curses1.py » et exécutez-le dans un terminal. Quelques remarques : quand vous utilisez une bordure, elle occupe une des positions disponibles pour chaque caractère de la bordure. De plus, les numéros de lignes et de positions (colonnes) commencent tous les deux à ZÉRO. Cela signifie que la première ligne de notre écran est la ligne 0, et la dernière ligne est la ligne 23. Ainsi, la position en haut à gauche est désignée par 0,0 et la position en bas à droite par 23,79. Créons un exemple rapide pour démontrer cela (en haut à droite).

Exemple très simple, si ce n'est le

bloc try/finally. Rappelez-vous, j'ai dit qu'il était TRÈS important d'appeler curses.endwin avant la fin de votre programme. De cette manière, même si les choses tournent mal, la routine endwin sera appelée. Il y a plusieurs façons d'aboutir à ce résultat, mais celle-ci me semble assez simple.

Créons maintenant un joli système de menu. Rappelez-vous, il y a quelque temps, nous avons écrit une application de gestion de recettes de cuisine qui avait un menu (dans la partie 8 de cette série d'articles). Tout défilait dans le terminal lorsque nous affichions quelque chose. Cette fois-ci, nous reprendrons cette idée et ferons un patron de menu que vous pourrez utiliser pour améliorer l'application de la partie 8. Ci-dessous, vous trouverez ce que nous avions écrit cette fois-là.

Cette fois-ci, nous utiliserons Curses. Commençons avec le patron suivant. Vous pouvez sauvegarder ce morceau de code (en bas à droite) pour pouvoir le réutiliser dans vos futurs programmes. Maintenant, sauvez à nouveau ce morceau de code sous le nom « menucurses1.py » pour pouvoir travailler sur ce fichier et garder l'original intact.

Avant d'aller plus loin avec notre code, nous allons travailler de façon modulaire. Voici (en haut à droite) un exemple de ce que nous allons faire, écrit en pseudo-code.

Bien entendu, ce pseudo-code n'est que ça : pseudo. Mais cela vous donne une idée de notre objectif avec tout ça. Puisqu'il ne s'agit que d'un exemple, nous allons nous arrêter là, mais vous pouvez le continuer si vous voulez. Commençons avec la boucle principale (page suivante, au milieu, à droite).

```
#!/usr/bin/env python
# CursesExample2
import curses
#================================================================
# BOUCLE PRINCIPALE
#================================================================
try:
    monEcran = curses.initscr()
    monEcran.clear()
    monEcran.addstr(0,0,"0       1        2        3
       4          5          6          7")
monEcran.addstr(1,0,"12345678901234567890123456789012345678
90123456789012345678901234567890123456789012345678901234567890")
    monEcran.addstr(10,0,"10")
    monEcran.addstr(20,0,"20")
    monEcran.addstr(23,0, "23 - Appuyez sur une touche
pour continuer")
    monEcran.refresh()
    monEcran.getch()
finally:
    curses.endwin()
```

```
#!/usr/bin/env python
#---------------------------------
# Modèle de programmation de Curses
#---------------------------------
import curses
def InitScreen(Border):
    if Border == 1:
        myscreen.border(0)
#================================================================
# BOUCLE PRINCIPALE
#================================================================
myscreen = curses.initscr()
InitScreen(1)
try:
    myscreen.refresh()
    # Votre code ici...
    myscreen.addstr(1,1, "Appuyez sur une touche pour
continuer")
    myscreen.getch()
finally:
    curses.endwin()
```

```
==================================================
BASE DE DONNEES DE RECETTES
==================================================
1 - Afficher toutes les recettes
2 - Rechercher une recette
3 - Afficher une recette
4 - Supprimer une recette
5 - Ajouter une recette
6 - Imprimer une recette
0 - Quitter
==================================================
Saisissez votre choix ->
```

Pas beaucoup de programmation ici. Nous avons notre bloc try/finally comme dans notre exemple. On initialise l'écran Curses puis on appelle la routine BouclePrincipale. Ce code est en bas, tout à droite.

Encore une fois, peu de choses ici, mais il ne s'agit que d'un exemple. Ici on appelle deux routines, l'une est AfficherMenuPrincipal et l'autre est SaisieMenuPrincipal. AfficherMenuPrincipal (voir en bas de la page précédente) affichera notre menu principal et SaisieMenuPrincipal s'occupe de gérer ce menu.

Notez que cette routine ne fait qu'effacer l'écran (monEcran.erase) puis affiche ce que nous voulons sur l'écran. Il n'y a ici aucun traitement des saisies clavier, c'est le boulot de la routine SaisieMenuPrincipal que vous pouvez voir ci-dessous.

C'est vraiment une routine simple. On saute dans une boucle « while » jusqu'à ce que l'utilisateur appuie sur la touche 0. Dans cette boucle, on vérifie si cette touche est égale à différentes valeurs et, si c'est le cas, on appelle une série de routines et, finalement, on appelle le menu principal quand on a terminé. Vous pouvez compléter la plupart de ces routines

```
curses.initscreen
LogicLoop
    ShowMainMenu                        # affiche le menu principal
    MainInKey                           # voici la routine principale de saisie
        While Key != 0:
            If Key == 1:
                ShowAllRecipesMenu      # affiche le menu Toutes les recettes
                Inkey1                  # traite les saisies pour ce menu
                ShowMainMenu            # affiche le menu principal
            If Key == 2:
                SearchForARecipeMenu    # affiche le menu Rechercher une recette
                InKey2                  # traite les saisies pour ce menu
                ShowMainMenu            # affiche le menu principal
            If Key == 3:
                ShowARecipeMenu         # affiche le menu Affiche une recette
                InKey3                  # traite les saisies pour ce menu
                ShowMainMenu            # affiche le menu principal
                ...                     # et ainsi de suite
curses.endwin()                         # Rétablit le terminal
```

```
def AfficherMenuPrincipal():
  monEcran.erase()
  monEcran.addstr(1,1,
"========================================")
  monEcran.addstr(2,1, "
  Base de donnees de
recettes")
  monEcran.addstr(3,1,
"========================================")
  monEcran.addstr(4,1, " 1 - Voir toutes les
recettes")
  monEcran.addstr(5,1, " 2 - Rechercher une
recette")
  monEcran.addstr(6,1, " 3 - Afficher une recette")
  monEcran.addstr(7,1, " 4 - Supprimer une
recette")
  monEcran.addstr(8,1, " 5 - Ajouter une recette")
  monEcran.addstr(9,1, " 6 - Imprimer une recette")
  monEcran.addstr(10,1, " 0 - Quitter")
  monEcran.addstr(11,1,
"========================================")
  monEcran.addstr(12,1, " Saisissez votre choix : ")
  monEcran.refresh()
```

```
# Boucle principale
try:
  monEcran = curses.initscr()
  BouclePrincipale()
finally:
  curses.endwin()
```

```
def BouclePrincipale():
  AfficherMenuPrincipal()
  SaisieMenuPrincipal()
```

par vous-même maintenant, mais nous allons regarder à présent l'option 2, Rechercher une recette. Le menu est court et simple. La routine SaisieMenu2 (ci-contre à droite) est un peu plus compliquée.

Nous utilisons à nouveau une boucle « while » standard ici. On règle la variable faireboucle à 1 pour obtenir une boucle sans fin jusqu'à obtenir ce qu'on veut. On utilise la commande break pour sortir de cette boucle. Les trois options sont très ressemblantes, la principale différence étant qu'on commence avec une variable tmpstr à laquelle on concatène tel ou tel texte selon ce qu'on aura sélectionné, rendant les choses un peu plus agréables. On appelle ensuite une routine RecupererTexteRecherche pour récupérer la chaîne à rechercher. On utilise la routine getstr pour récupérer une chaîne saisie par l'utilisateur plutôt qu'un seul caractère. Ensuite, on renvoie cette chaîne à notre routine de saisie pour qu'elle soit utilisée.

```
def SaisieMenuPrincipal():
  touche = 'X'
  while touche != ord('0'):
      touche = monEcran.getch(12,27)
      monEcran.addch(12,22,touche)
      if touche == ord('1'):
          MenuVoirToutesLesRecettes()
          AfficherMenuPrincipal()
      elif touche == ord('2'):
          MenuRechercherUneRecette()
          SaisieMenu2()
          AfficherMenuPrincipal()
      elif touche == ord('3'):
          MenuAfficherUneRecette()
          AfficherMenuPrincipal()
      elif touche == ord('4'):
          PasPrete("'Supprimer une recette'")
          AfficherMenuPrincipal()
      elif touche == ord('5'):
          PasPrete("'Ajouter une recette'")
          AfficherMenuPrincipal()
      elif touche == ord('6'):
          PasPrete("'Imprimer une recette'")
          AfficherMenuPrincipal()
      monEcran.refresh()
```

```
def MenuRechercherUneRecette():
  monEcran.addstr(4,1, "------------------------------")
  monEcran.addstr(5,1, " Rechercher dans")
  monEcran.addstr(6,1, "------------------------------")
  monEcran.addstr(7,1, " 1 - Nom de la recette")
  monEcran.addstr(8,1, " 2 - Source de la recette")
  monEcran.addstr(9,1, " 3 - Ingredients")
  monEcran.addstr(10,1," 0 - Quitter")
  monEcran.addstr(11,1,"Entrez le type de recherche -> ")
  monEcran.refresh()

def SaisieMenu2():
  touche = 'X'
  faireBoucle = 1
  while faireBoucle == 1:
      touche = monEcran.getch(11,32)
      monEcran.addch(11,22,touche)
      tmpstr = "Entrez le texte a rechercher dans "
      if touche == ord('1'):
          sstr = "'le nom de la recette' -> "
          tmpstr = tmpstr + sstr
          retstring = RecupererTexteRecherche(13,1,tmpstr)
          break
      elif touche == ord('2'):
          sstr = "'la source de la recette' -> "
          tmpstr = tmpstr + sstr
          retstring = RecupererTexteRecherche(13,1,tmpstr)
          break
      elif touche == ord('3'):
          sstr = "'les ingredients' -> "
          tmpstr = tmpstr + sstr
          retstring = RecupererTexteRecherche(13,1,tmpstr)
          break
      else:
          retstring = ""
          break
  if retstring != "":
      monEcran.addstr(15,1,"Vous avez saisi - " + retstring)
  else:
      monEcran.addstr(15,1,"Vous avez saisi une chaine vide")
  monEcran.refresh()
  monEcran.addstr(20,1,"Appuyez sur une touche")
  monEcran.getch()
def RecupererTexteRecherche(row,col,strng):
  monEcran.addstr(row,col,strng)
  monEcran.refresh()
  instring = monEcran.getstr(row,len(strng)+1)
  monEcran.addstr(row,len(strng)+1,instring)
  monEcran.refresh()
  return instring
```

Le code complet est ici : http://pastebin.com/dRHM8sre

Une dernière chose : si vous voulez aller plus loin avec la programmation Curses, il existe de nombreuses autres méthodes que celles que nous avons utilisées ce mois-ci. À part une recherche Google, un bon point de départ est la documentation officielle : http://docs.python.org/library/curses.html.

À la prochaine fois.

Podcast Full Circle

AUDIO MP3 AUDIO OGG

Le ***Podcast Full Circle*** est de retour et meilleur que jamais !

Les thèmes de cet épisode sont :
• Actualités.
• Opinion : contribuer à des articles avec le rédacteur en chef de FCM.
• Interview avec Amber Graner.
• Retours.
... et toute la bonne humeur habituelle.

Vos animateurs :
Robin Catling
Ed Hewitt
Ronnie Tucker

Le podcast et les notes sur l'émission sont visibles ici : http://fullcirclemagazine.org/

TUTORIEL Programmer en Python - Partie 14

Greg Walters

La dernière fois, nous avons parlé de la bibliothèque Curses. Cette fois-ci nous allons approfondir notre connaissance de cette bibliothèque et nous concentrer sur les commandes de couleurs. Si vous avez raté le dernier article, faisons un rappel rapide. Tout d'abord, il faut importer la bibliothèque curses ; puis appeler curses.initscr pour démarrer le programme. Pour afficher du texte à l'écran on appelle la fonction addstr, puis la fonction refresh pour faire apparaître les changements à l'écran. Enfin, il faut appeler curses.endwin() pour rendre son état initial à la fenêtre du terminal.

Maintenant, nous allons créer un programme rapide et facile qui utilise les couleurs. Ça ressemble beaucoup à ce que nous avons fait la dernière fois, mais nous allons voir quelques nouvelles commandes. Tout d'abord, on utilise curses.start_color() pour dire au système que nous voulons utiliser des couleurs dans notre programme. Puis on assigne une paire de couleurs de premier plan et d'arrière-plan. On peut assigner plusieurs paires et les utiliser quand nous le souhaitons. On fait cela grâce à la fonction curses.init_pair dont la syntaxe est :

```
curses.init_pair([numéro de paire],[couleur de premier plan],[couleur d'arrière plan])
```

On règle les couleurs en utilisant curses.COLOR_ suivi de la couleur que l'on souhaite. Par exemple, curses.COLOR_BLUE ou curses.COLOR_GREEN. Les options sont black (noir), red (rouge), green (vert), yellow (jaune), blue (bleu), magenta, cyan et white (blanc), à ajouter en majuscules à la suite de « curses.COLOR_ ». Une fois qu'on a réglé une paire de couleurs, on peut l'utiliser comme dernier paramètre de la fonction screen.addstr ainsi :

```
myscreen.addstr([ligne],[colonne],[texte],curses.color_pair(X))
```

où X est la paire de couleurs que l'on souhaite utiliser.

Sauvegardez le code suivant (ci-dessus à droite) dans le fichier testcouleur1.py et exécutez-le. N'essayez pas de lancer un programme curses dans un environnement de développement comme SPE ou Dr Python ; exécutez-le dans un terminal.

```
import curses
try:
    monecran = curses.initscr()
    curses.start_color()
    curses.init_pair(1, curses.COLOR_BLACK, curses.COLOR_GREEN)
    curses.init_pair(2, curses.COLOR_BLUE, curses.COLOR_WHITE)
    curses.init_pair(3, curses.COLOR_MAGENTA,curses.COLOR_BLACK)
    monecran.clear()
    monecran.addstr(3,1,"  Ceci est un test ",curses.color_pair(1))
    monecran.addstr(4,1,"  Ceci est un test ",curses.color_pair(2))
    monecran.addstr(5,1,"  Ceci est un test ",curses.color_pair(3))
    monecran.refresh()
    monecran.getch()
finally:
    curses.endwin()
```

Vous devriez voir un fond gris, avec trois lignes de texte disant « Ceci est un test » dans différentes couleurs. La première devrait être noir sur vert, la deuxième bleu sur blanc, et la troisième magenta sur noir.

Souvenez-vous du bloc try/finally. Il permet au programme de remettre le terminal en bon état automatiquement si jamais quelque chose se passe de travers. Il existe une autre façon de faire : curses fournit une fonction nommée wrapper. Wrapper fait tout le travail pour vous : elle appelle curses.initscr(), curses.start_color() et curses.endwin() à votre place. La seule chose dont vous devez vous

souvenir est d'appeler la fonction wrapper avec la fonction « main » en argument. Cela envoie un pointeur sur votre écran. Sur la page suivante (en haut à droite) vous verrez le même programme que le précédent mais qui utilise la fonction curses.wrapper.

C'est bien plus simple et nous n'avons pas à nous préoccuper d'appeler curses.endwin() si quelque chose se passe mal. Tout le boulot est fait automatiquement.

Maintenant que nous avons acquis les bases, mettons en œuvre ce que nous avons déjà appris et commençons à programmer un jeu. Cependant, avant de démarrer, planifions ce que nous allons faire. Notre jeu va choisir au hasard une lettre majuscule, la déplacer de la droite de l'écran vers la gauche ; puis à une position aléatoire la lettre tombera vers le bas de l'écran. Nous aurons un canon qui peut être déplacé avec les flèches droite et gauche du clavier pour le placer en dessous de la lettre qui tombe ; en appuyant sur la barre d'espace, le canon pourra tirer. Si on touche la lettre avant qu'elle n'arrive en bas de l'écran, on gagne un point ; sinon, notre canon explose. Si on perd trois canons, la partie est finie. Bien que cela ait l'air assez simple, coder ce jeu nécessite quand même beaucoup de travail.

Commençons. Il faut initialiser le jeu, et créer quelques routines avant d'aller plus loin. Créez un nouveau projet nommé jeu1.py ; commencez avec le code ci-contre, en bas à droite.

Ce code ne fait pas grand chose pour l'instant, mais ce n'est que le début. Notez que l'on a quatre instructions init_pair pour régler les couleurs que nous utiliserons pour nos ensembles de couleurs aléatoires, et une pour les explosions (l'ensemble numéro 5). Maintenant il nous faut régler des variables et des constantes qui seront utilisées pendant le jeu. Nous les mettrons dans la routine init de la classe Jeu1. Remplacez les instructions « pass » dans init par le code de la page suivante.

Vous devriez être à même de comprendre ce qui se passe dans ces définitions. Si vous n'êtes pas sûr pour le moment, ça devrait devenir plus clair lorsque nous compléterons le code.

Nous approchons de quelque chose qui va tourner. Il nous reste encore quelques routines à construire avant

```
import curses
def main(ecran):
    curses.init_pair(1, curses.COLOR_BLACK,
curses.COLOR_GREEN)
    curses.init_pair(2, curses.COLOR_BLUE,
curses.COLOR_WHITE)
    curses.init_pair(3,
curses.COLOR_MAGENTA,curses.COLOR_BLACK)
    ecran.clear()
    ecran.addstr(3,1,"  Ceci est un test
",curses.color_pair(1))
    ecran.addstr(4,1,"  Ceci est un test
",curses.color_pair(2))
    ecran.addstr(5,1,"  Ceci est un test
",curses.color_pair(3))
    ecran.refresh()
    ecran.getch()
curses.wrapper(main)
```

```
import curses
import random

class Jeu1():
    def __init__(self):
        pass
    def main(self,ecran):
        curses.init_pair(1, curses.COLOR_BLACK,
curses.COLOR_GREEN)
        curses.init_pair(2, curses.COLOR_BLUE,
curses.COLOR_BLACK)
        curses.init_pair(3, curses.COLOR_YELLOW,
curses.COLOR_BLUE)
        curses.init_pair(4, curses.COLOR_GREEN,
curses.COLOR_BLUE)
        curses.init_pair(5, curses.COLOR_BLACK,
curses.COLOR_RED)

    def Demarrage(self):
        curses.wrapper(self.main)
g = Jeu1()
g.Demarrage()
```

qu'il en fasse plus. Examinons la routine qui déplace une lettre de droite à gauche sur l'écran : http://fullcirclemagazine.pastebin.com/zct2nsni.

C'est la routine la plus longue de notre programme et elle contient de nouvelles fonctions. La fonction ecran.delch efface le caractère situé à la ligne et colonne indiquées. curses.napms() indique à Python de « dormir » [Ndt_: nap = sieste] pendant X millisecondes (ms).

Le fonctionnement logique de cette routine est expliqué en pseudo-code sur la page 29 (en haut à droite).

Vous devriez pouvoir suivre le code maintenant. Nous avons besoin de deux nouvelles routines pour faire les choses correctement. La première s'appelle Explose, et on la remplit avec l'instruction « pass ». La seconde s'appelle Reinitialise. C'est ici que nous réinitialiserons la ligne et la colonne courantes aux valeurs par défaut, remettrons à 0 le drapeau UneLettreTombe, choisirons une lettre et un point de chute au hasard. Ces deux routines se trouvent au milieu à droite de la page suivante.

Maintenant nous avons besoin de quatre autres routines pour continuer (en bas à droite de la page suivante). L'une choisit une lettre au hasard, l'autre choisit un point de chute au hasard. Rappelez-vous que nous avons déjà parlé du module « random » [Ndt : aléatoire] auparavant dans cette série.

```
        # ce qui suit concerne les lignes
         self.LigneCanon = 22              # ligne ou se trouve le canon
         self.PositionCanon = 39           # position ou le canon demarre
         self.LigneLettre = 2              # ligne ou les lettres passent de droite a gauche
         self.LigneScore = 1               # ligne ou se trouve le score
         self.PositionScore = 50           # position horizontale du score
         self.PositionVies = 65            # position horizontale des vies

         # ce qui suit concerne les lettres
         self.LettreActuelle = "A"         # variable contenant les lettres
         self.PositionLettreActuelle = 78 # position horizontale de depart des lettres
         self.PositionChute = 10           # position ou tombent les lettres
         self.UneLettreTombe = 0           # drapeau indiquant si les lettres tombent
         self.LigneLettreActuelle = 3      # ligne actuelle des lettres
         self.CompteurLettres = 15         # combien de boucles avant de retourner  travailler ?

         # ce qui suit concerne les tirs
         self.CanonTire = 0                # drapeau : est-ce que le canon tire ?
         self.LigneTir = self.LigneCanon - 1
         self.ColonneTir = self.PositionCanon

         # autres informations
         self.CompteurBoucles = 0          # compte le nombre de boucles
         self.Score = 0                    # score actuel
         self.Vies = 3                     # nombre de vies par defaut
         self.CouleurActuelle = 1          # couleur actuelle
         self.DiminuerScoreSiEchec = 0     # regler a 1 pour decrementer le score
                                           # lorsqu'une lettre touche le bas
```

Dans ChoisirUneLettre, on génère un nombre aléatoire entre 65 et 90 (le code des lettres de A à Z). Rappelez-vous que pour utiliser la fonction de tirage aléatoire on doit lui fournir une plage de nombres, à savoir un minimum et un maximum. Il se passe la même chose dans ChoisirPointDeChute. Dans les deux routines, nous appelons la fonction random.seed() qui règle le générateur de nombres aléatoires de façon différente à chaque fois qu'elle est appelée. La troisième routine s'appelle

VerifierTouches ; elle examine chaque touche du clavier sur lequel l'utilisateur appuie et s'en sert pour déplacer le canon. Nous la laisserons de côté pour le moment, mais nous en aurons besoin plus tard. Nous aurons également besoin de la routine VerifieCollision, que nous laissons aussi de côté pour l'instant.

```
def
VerifierTouches(self,ecran,saisie):
    pass
def
VerifieCollision(self,ecran):
    pass
```

On va créer une petite routine, appelée BoucleDeJeu, qui sera le « cerveau » de notre jeu (en haut à droite de la page 30).

La logique de cette routine est de régler notre clavier à nodelay(1), ce qui signifie que nous n'attendrons pas de saisie clavier et que, lorsqu'il y a une saisie, on la met en cache pour la traiter plus tard. Puis on entre dans une boucle while infinie (la condition est toujours vraie car égale à 1), ce qui signifie que le jeu continue jusqu'à ce qu'on soit prêt à l'arrêter. On dort pendant 40 millisecondes, on déplace la lettre, puis on vérifie si l'utilisateur a appuyé sur une touche. Si c'est un « Q » (notez que c'est une majuscule), ou bien la touche ESC, alors on sort de la boucle pour terminer le programme. Sinon, on vérifie si c'est la flèche gauche ou droite, ou la barre d'espace. Plus tard, vous pourrez rendre le jeu un peu plus difficile en vérifiant si la touche pressée est la même que la lettre affichée et en ne tirant que dans ce cas, comme dans un logiciel d'apprentissage du clavier. Souvenez-vous juste d'enlever le Q en tant que touche qui sert à quitter le jeu.

Nous aurons également besoin de créer une routine qui initialise chaque

```
SI on a attendu le bon nombre de boucles ALORS
   remettre à 0 le compteur de boucles
   SI on bouge vers la gauche de l'écran ALORS
     effacer le caractère à la ligne et colonne courantes
     attendre 50 millisecondes
     SI la colonne courante est supérieure à 2 ALORS
          décrémenter la colonne courante
     placer le caractère à la ligne et colonne courantes
     SI la colonne courante est égale à la colonne aléatoire pour faire tomber la
lettre ALORS
          régler le drapeau UneLettreTombe à 1
   SINON
     effacer le caractère à la ligne et colonne courantes
     attendre 50 millisecondes
     SI la ligne courante est inférieure à la ligne où se trouve le canon ALORS
          incrémenter la ligne courante
          placer le caractère à la ligne et colonne courantes
     SINON
          Explose (et décrémentez le score si vous le souhaitez) et vérifier si on
continue
          choisir une nouvelle lettre et une nouvelle position et tout recommencer
SINON
   incrémenter le compteur de boucles
rafraîchir l'écran
```

```
    def Explose(self,ecran):
        pass
    def Reinitialise(self):
        self.LigneLettreActuelle = self.LigneLettre
        self.PositionLettreActuelle = 78
        self.UneLettreTombe = 0
        self.ChoisirUneLettre()
        self.ChoisirPointDeChute()
```

```
   def ChoisirUneLettre(self):
        random.seed()
        lettre = random.randint(65,90)
        self.LettreActuelle = chr(lettre)

    def ChoisirPointDeChute(self):
        random.seed()
        self.PositionChute = random.randint(3,78)
```

nouvelle partie. Appelons-la NouvellePartie (ci-contre, au milieu à droite).

Nous avons également besoin de la routine AfficheScore qui montre le score actuel et le nombre de vies restantes (ci-contre, en bas à droite).

Maintenant il nous reste à ajouter du code à notre routine principale (ci-dessous, à gauche) pour démarrer la boucle de jeu. Le code supplémentaire est en dessous, ajoutez-le sous le dernier appel à init_pair.

Nous avons maintenant un programme qui fait quelque chose. Essayez-le, je vous attends.

Nous avons maintenant un programme qui choisit au hasard une lettre majuscule, la déplace de la droite de l'écran vers la gauche sur un nombre aléatoire de colonnes, puis déplace cette lettre vers le bas de l'écran. Cependant, la première chose que vous devez remarquer est que, quand vous lancez le programme, la première lettre est toujours un « A », et le point de chute est toujours à la colonne 10. C'est parce qu'on règle des valeurs par défaut dans la routine init. Pour réparer ça, appelez simplement self.Reinitialise

```
        ecran.addstr(11,28,"Bienvenue dans l'attaque des
lettres")
        ecran.addstr(13,28,"Appuyez sur une touche pour
commencer...")
        ecran.getch()
        ecran.clear()
        BoucleDeJeu = 1
        while BoucleDeJeu == 1:
            self.NouvellePartie(ecran)
            self.BoucleDeJeu(ecran)
            ecran.nodelay(0)
            curses.flushinp()
            ecran.addstr(11,35,"Fin de la partie")
            ecran.addstr(13,23,"Voulez-vous rejouer ?
(O/N)")
            saisie = ecran.getch(14,56)
            if saisie == ord("N") or saisie == ord("n"):
                break
            else:
                ecran.clear()
```

```
    def BoucleDeJeu(self,ecran):
        test = 1              # gere la boucle
        while test == 1:
            curses.napms(20)
            self.BougeLettre(ecran)
            saisie =
ecran.getch(self.LigneScore,self.PositionScore)
            if saisie == ord('Q') or saisie == 27:  # 'Q'
ou <Esc>
                break
            else:
                self.VerifierTouches(ecran,saisie)
            self.AfficheScore(ecran)
            if self.Vies == 0:
                break
        curses.flushinp()
        ecran.clear()
```

```
    def NouvellePartie(self,ecran):
        self.CaractereCanon = curses.ACS_SSBS

ecran.addch(self.LigneCanon,self.PositionCanon,self.Cara
ctereCanon,curses.color_pair(2) | curses.A_BOLD)
        ecran.nodelay(1)    # on n'attend pas de saisie
clavier
        self.Reinitialise()
        self.Score = 0
        self.Vies = 3
        self.AfficheScore(ecran)
        ecran.move(self.LigneScore,self.PositionScore)
```

```
    def AfficheScore(self,ecran):

ecran.addstr(self.LigneScore,self.PositionScore,"SCORE :
%d" % self.Score)

ecran.addstr(self.LigneScore,self.PositionVies,"VIES :
%d" % self.Vies)
```

avant d'entrer dans la boucle while de la routine principale.

Maintenant, nous devons travailler sur les routines qui gèrent notre canon. Ajoutez à la classe Jeu1 le code situé ci-contre, en haut à droite.

BougeCanon prend la position courante du canon et le déplace dans la direction où on veut qu'il aille. La seule chose nouvelle dans cette routine est située à la fin de la fonction addch. On appelle la paire de couleurs (2) pour régler la couleur et, en même temps, on force le canon à s'afficher en gras. On utilise un « ou bit à bit » (« | ») pour forcer l'attribut. Puis on doit étoffer notre routine VerifierTouches : remplacez l'instruction « pass » avec le nouveau code (ci-contre, au milieu à droite).

Maintenant il faut écrire une routine pour déplacer la balle qui va exploser vers le haut de l'écran (en haut à droite de la page suivante).

On a encore besoin de quelques routines (ci-contre, en bas à droite) avant d'en avoir terminé. Voici le code pour remplir la routine VerifieCollision et le code pour TirExplose.

Enfin nous devons étoffer notre routine Explose : remplacez « pass » par le code situé en haut de la page suivante.

Nous avons maintenant un programme qui fonctionne. Vous pouvez régler la valeur de CompteurLettre pour accélérer ou ralentir le mouvement de la lettre qui traverse l'écran pour rendre le jeu plus ou moins facile. Vous pouvez également utiliser la variable CouleurActuelle pour faire un choix de couleur aléatoire et régler la couleur de la lettre sur l'un des quatre ensembles de couleurs que nous avons créés et changer la façon dont la couleur est réglée. Je voulais vous lancer un défi.

```
    def BougeCanon(self,ecran,direction):
        ecran.addch(self.LigneCanon,self.PositionCanon," ")
        if direction == 0:     # gauche
            if self.PositionCanon > 0:
                self.PositionCanon -= 1
        elif direction == 1:  # droite
            if self.PositionCanon < 79:
                self.PositionCanon += 1

ecran.addch(self.LigneCanon,self.PositionCanon,self.Carac
tereCanon,curses.color_pair(2) | curses.A_BOLD)
```

```
        if saisie == 260: # fleche a gauche (pas sur le pave numerique)
            self.BougeCanon(ecran,0)
            curses.flushinp()  # vide le tampon clavier
        elif saisie == 261: # fleche a droite (pas sur le pave numerique)
            self.BougeCanon(ecran,1)
            curses.flushinp()  # vide le tampon clavier
        elif saisie == 52:  # fleche a gauche sur le pave numerique
            self.BougeCanon(ecran,0)
            curses.flushinp()  # vide le tampon clavier
        elif saisie == 54:  # fleche a droite sur le pave numerique
            self.BougeCanon(ecran,1)
            curses.flushinp()  # vide le tampon clavier
        elif saisie == 32:  # espace
            if self.CanonTire == 0:
                self.CanonTire = 1
                self.ColonneTir = self.PositionCanon
                ecran.addch(self.LigneTir,self.ColonneTir,"|")
                curses.flushinp()  # vide le tampon clavier
```

```
    def BougeTir(self,ecran):
        ecran.addch(self.LigneTir,self.ColonneTir," ")
        if self.LigneTir > self.LigneLettre:
            self.VerifieCollision(ecran)
            self.LigneTir -= 1
            ecran.addch(self.LigneTir,self.ColonneTir,"|")
        else:
            self.VerifieCollision(ecran)
            ecran.addch(self.LigneTir,self.ColonneTir," ")
            self.LigneTir = self.LigneCanon - 1
            self.CanonTire = 0
```

J'espère que vous vous êtes amusés cette fois-ci et que vous ajouterez des fonctionnalités pour rendre le jeu plus agréable. Comme d'habitude, le code complet se trouve sur www.thedesignatedgeek.com, ou bien ici : http://fullcirclemagazine.pastebin.com/jaNZSvkg.

```
    def VerifieCollision(self,ecran):
        if self.CanonTire == 1:
            if self.LigneTir == self.LigneLettreActuelle:
                if self.ColonneTir == self.PositionLettreActuelle:
                    ecran.addch(self.LigneTir,self.ColonneTir," ")
                    self.TirExplose(ecran)
                    self.Score +=1
                    self.Reinitialise()

    def TirExplose(self,ecran):
        ecran.addch(self.LigneTir,self.ColonneTir,"X",curses.color_pair(5))
        ecran.refresh()
        curses.napms(200)
        ecran.addch(self.LigneTir,self.ColonneTir,"|",curses.color_pair(5))
        ecran.refresh()
        curses.napms(200)
        ecran.addch(self.LigneTir,self.ColonneTir,"-",curses.color_pair(5))
        ecran.refresh()
        curses.napms(200)
        ecran.addch(self.LigneTir,self.ColonneTir,".",curses.color_pair(5))
        ecran.refresh()
        curses.napms(200)
        ecran.addch(self.LigneTir,self.ColonneTir," ",curses.color_pair(5))
        ecran.refresh()
        curses.napms(200)
```

```
        ecran.addch(self.LigneLettreActuelle,self.PositionLettreActuelle,"X",curses.color_pair(5))
        curses.napms(100)
        ecran.refresh()
        ecran.addch(self.LigneLettreActuelle,self.PositionLettreActuelle,"|",curses.color_pair(5))
        curses.napms(100)
        ecran.refresh()
        ecran.addch(self.LigneLettreActuelle,self.PositionLettreActuelle,"-",curses.color_pair(5))
        curses.napms(100)
        ecran.refresh()
        ecran.addch(self.LigneLettreActuelle,self.PositionLettreActuelle,".",curses.color_pair(5))
        curses.napms(100)
        ecran.refresh()
        ecran.addch(self.LigneLettreActuelle,self.PositionLettreActuelle," ")
        ecran.addch(self.LigneCanon,self.PositionCanon,self.CaractereCanon,curses.color_pair(2) | curses.A_BOLD)
ecran.refresh()
```

TUTORIEL Programmer en Python - Partie 15

Greg Walters

Ce mois-ci, nous allons explorer Pygame, un ensemble de modules conçu pour écrire des jeux. Le site web est : http://www.pygame.org/. Pour reprendre le fichier Lisez-moi de Pygame : « Pygame est une bibliothèque multiplateforme conçue pour faciliter l'écriture de logiciels multimédia, comme les jeux en Python. Pygame requiert le langage Python et la bibliothèque multimédia SDL, mais il peut aussi utiliser plusieurs bibliothèques populaires. »

Vous pouvez installer Pygame par Synaptic, avec le paquet « python-pygame ». Faites cela maintenant pour qu'on puisse aller plus loin.

Tout d'abord, nous importons Pygame (voir ci-dessus à droite). Puis nous réglons os.environ pour centrer la fenêtre sur l'écran. Ensuite, nous initialisons Pygame, puis réglons sa fenêtre à une taille de 800x600 pixels et lui donnons un titre. Pour finir, nous affichons l'écran et entrons dans une boucle en attendant la frappe d'une touche au clavier ou le clic sur un bouton de la souris. L'écran est un objet qui contiendra tout ce qu'on décide d'y placer. On l'appelle une surface. Imaginez-le comme une feuille de papier sur laquelle nous allons dessiner des choses.

Pas très excitant, mais c'est un début. Rendons les choses un peu moins ennuyeuses. On peut changer la couleur de fond en quelque chose de moins sombre. J'ai trouvé un programme appelé « colorname » que vous pouvez installer à partir de la logithèque Ubuntu. Il vous permet d'utiliser une roue des couleurs pour choisir la couleur qui vous plaît et vous donnera les valeurs RVB (rouge, vert, bleu) pour cette couleur. On doit passer par les couleurs RVB si on ne veut pas utiliser celles prédéfinies fournies par Pygame. C'est un utilitaire sympa que vous devriez songer à installer.

Juste après les instructions d'importation, ajoutez :

```
couleurFond = 208, 202, 104
```

Ceci réglera la variable couleurFond à une couleur un peu terre de Sienne. Puis, après la ligne pygame.display.set_caption, ajoutez les lignes suivantes :

```
# voici les import
import pygame
from pygame.locals import *
import os
# pour centrer le jeu sur l'ecran
os.environ['SDL_VIDEO_CENTERED'] = '1'
# initialise Pygame
pygame.init()
# initialise l'ecran
ecran = pygame.display.set_mode((800, 600))
# regle le caption (barre de titre de la fenetre)
pygame.display.set_caption('Pygame Test #1')
# affiche l'ecran et attend un evenement
faireBoucle = 1
while faireBoucle:
    if pygame.event.wait().type in (KEYDOWN,
MOUSEBUTTONDOWN):
        break
```

```
ecran.fill(couleurFond)
pygame.display.update()
```

La méthode ecran.fill() réglera la couleur à ce qu'on lui passe en argument. La ligne suivante, pygame.display.update(), applique réellement les changements à l'écran.

Sauvez tout cela sous le nom pygame1.py et continuons.

Maintenant, affichons du texte dans notre fenêtre un peu vide. À nouveau, commençons par les instructions d'importation et le réglage de la variable de couleur du fond du programme précédent :

```
import pygame
from pygame.locals import *
import os
couleurFond = 208, 202, 104
```

Maintenant, ajoutons une variable supplémentaire pour la couleur de premier plan pour notre police :

```
couleurPolice = 255,255,255
# blanc
```

Puis nous ajoutons la plus grande partie du code de l'exemple précédent (voir à droite).

Si vous exécutez cela maintenant, rien n'a apparemment changé puisque tout ce que nous avons fait est d'ajouter la définition de la couleur de police. Maintenant, après la ligne ecran.fill() et avant la partie concernant la boucle, saisissez les lignes suivantes :

```
police =
pygame.font.Font(None,27)
texte = police.render('Voici
du texte', True,
couleurPolice, couleurFond)
texte_rect =
texte.get_rect()
ecran.blit(texte,texte_rect)
pygame.display.update()
```

Allez, sauvez le programme sous le nom pygame2.py et exécutez-le. En haut à gauche de la fenêtre, vous devriez voir le texte « Voici du texte ».

Regardons de plus près les nouvelles commandes. D'abord on appelle la méthode « font » en lui passant deux arguments. Le premier est le nom de la police que nous voulons utiliser et le deuxième est la taille de la police. Pour le moment, nous utilisons « None » pour laisser le système choisir une police générique à notre place, et on règle la taille à 27.

Ensuite, nous avons la méthode police.render(). Elle prend quatre arguments qui sont, dans l'ordre : le texte à afficher, l'utilisation ou non de l'antialiasing (True pour vrai), la couleur de premier plan de la police et enfin sa couleur de fond.

La ligne suivante (texte.get_rect() récupère un objet de type rectangle que nous utiliserons pour placer le texte sur l'écran. Ceci est important, car presque tout ce que nous allons faire ensuite va se passer avec des rectangles (vous en saurez plus dans un instant). Puis on « blit » le rectangle sur l'écran et, enfin, on rafraîchit ce dernier pour afficher le texte. Que signifie « blit » et pourquoi diable voudrais-je faire une chose qui sonne aussi bizarrement ? Eh bien, ce terme remonte aux années 1970 et vient du parc Xerox (à qui on doit de nombreuses technologies actuelles). Ce terme s'appelait au départ BitBLT qui signifie Bit (pour bitmap) et Block Transfert (ou transfert d'images par bloc) : puis il est devenu Blit (sans doute parce que c'est plus court). Simplement, cela signifie qu'on fait apparaître une image ou un texte à l'écran.

Comment faire pour que le texte soit centré sur l'écran au lieu d'être placé sur la première ligne où on met du temps à la voir ? Entre la ligne text.get_rect() et la ligne ecran.blit,

```
# pour centrer le jeu sur l'ecran
os.environ['SDL_VIDEO_CENTERED'] = '1'
# initialise Pygame
pygame.init()
# initialise l'ecran
ecran = pygame.display.set_mode((800, 600))
# regle le caption (barre de titre de la fenetre)
pygame.display.set_caption('Pygame Test #1')
ecran.fill(Background)
pygame.display.update()

# notre boucle
faireBoucle = 1
while faireBoucle:
    if pygame.event.wait().type in
(KEYDOWN,MOUSEBUTTONDOWN):
        break
```

placez les deux lignes suivantes :

```
texte_rect.centerx =
ecran.get_rect().centerx
texte_rect.centery =
ecran.get_rect().centery
```

Elles servent à récupérer les coordonnées horizontales et verticales du centre de notre objet écran (souvenez-vous de la surface) en pixels et à régler les coordonnées x et y du centre de notre rectangle à cet endroit.

Exécutez le programme. Maintenant le texte est affiché au centre de la surface. Vous pouvez également modifier le texte en utilisant (dans notre exemple) police.set_bold(True) et/ou police.set_italic(True) juste après la ligne pygame.font.True.

Souvenez-vous que nous avons discuté rapidement de l'option « None » lorsque nous avons réglé la police générique. Disons que vous voulez utiliser maintenant une police plus jolie. Comme je l'ai dit précédemment, la méthode pygame.font.Font() prend deux arguments. Le premier est le chemin et le nom du fichier contenant la police à utiliser et le second est la taille de la police. Plusieurs questions se posent à ce stade. Comment connaît-on le chemin et le nom du fichier à utiliser pour la police sur un système quelconque donné ? Heureusement, Pygame fournit une fonction qui s'occupe de cela pour nous. Elle s'appelle match_font. Voici un court programme qui affiche le chemin et le nom du fichier de (c'est un exemple) la police « Courier New ».

```
import pygame
from pygame.locals import *
import os
print
pygame.font.match_font('Couri
er New')
```

Sur mon système la valeur affichée est « /usr/share/fonts/truetype/mdttcorefonts/cour.ttf ». Si la police n'est pas trouvée, la valeur retournée est « None ». En supposant que la police est réellement trouvée, alors on peut affecter la valeur retournée à une variable, puis l'utiliser dans l'instruction suivante :

```
courier =
pygame.font.match_font('Couri
er New')
police =
pygame.font.Font(courier,27)
```

Modifiez la dernière version de votre programme pour inclure ces deux lignes et exécutez-le à nouveau. L'essentiel est soit d'utiliser une police dont vous SAVEZ qu'elle existera sur la machine de l'utilisateur final, soit de l'inclure lorsque vous distribuez votre programme et de coder en dur le nom et le chemin de la police. Il existe d'autres façons de faire, mais je vous laisse chercher pour qu'on puisse continuer.

Le texte c'est bien, mais les graphismes sont encore mieux. J'ai trouvé un tutoriel pour Pygame qui est très bien fait, écrit par Peyton McCollugh, et j'ai pensé le réutiliser ici en le modifiant. Pour cet exemple, nous devons commencer par créer une image que nous allons ensuite déplacer sur notre surface.. Cette image est appelée un « fantôme » (ou « sprite » en anglais). Utilisez Gimp ou un autre logiciel pour créer un dessin. Rien de joli, simplement un dessin générique. Je vais supposer que vous utilisez Gimp. Créez une nouvelle image, réglez sa taille à 50 pixels, dans les deux dimensions, puis, dans les options avancées, réglez la couleur de remplissage à « transparent ». Utilisez l'outil « crayon » avec une brosse circulaire de taille 3. Dessinez votre croquis et sauvegardez-le sous le nom stick.png dans le même répertoire que le programme en Python. Voici à quoi ressemble mon dessin, je suis sûr que vous pouvez faire mieux.

Je sais... je ne suis pas un artiste, mais pour ce que nous allons en faire, cela fera l'affaire. Nous avons réalisé une image au format .png avec une couleur de fond transparente, de façon que seuls les traits noirs s'affichent, sans un fond blanc, ni d'une autre couleur.

```
import pygame
from pygame.locals import *
import os

couleurFond = 0,255,127
os.environ['SDL_VIDEO_CENTERED'] = '1'
pygame.init()
ecran = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Pygame exemple n° 4 - Fantome')
ecran.fill(couleurFond)
```

Parlons maintenant de ce que notre programme va faire. Nous voulons afficher une fenêtre Pygame qui contient notre dessin ; nous voulons déplacer le dessin avec les touches fléchées du clavier (haut, bas, droite, gauche), sauf si nous sommes au bord de l'écran auquel cas, on ne peut pas bouger plus loin. De plus, nous voulons quitter le jeu en appuyant sur la touche « q ». Déplacer le fantôme autour de la fenêtre peut sembler simple , et ça l'est, mais pas autant qu'on peut le croire au début. Commençons par créer deux rectangles, un pour le fantôme lui-même et un autre de la même taille, mais vide. Pour commencer, on affiche le fantôme sur la surface, puis, lorsque l'utilisateur appuie sur une touche, on affiche le rectangle vide par dessus le fantôme. C'est à peu près la même chose que ce que nous avons fait le mois dernier avec le jeu de l'alphabet. Nous avons à peu près terminé ce programme, qui nous donne un aperçu de ce qu'il faut faire pour afficher un dessin à l'écran et le déplacer.

Démarrez donc un nouveau programme, nommé pygame4.py. Placez-y les « include » que nous avons utilisés durant ce tutoriel. Cette fois-ci, nous utiliserons un fond vert menthe en prenant les valeurs 0, 255, 127 (voir ci-dessus). Puis on crée une classe qui se chargera du graphisme ou fantôme (page suivante en bas à gauche). Placez ce code juste après les « import ». Que fait tout cela ? Commençons par la routine __init__. On initialise le module de Pygame qui gère les fantômes avec la ligne pygame.sprite. Sprite.__init__. Puis on règle la surface et on la nomme « ecran », ce qui nous permettra de vérifier si le fantôme sort de l'écran. Ensuite on crée la variable « ancienFantome » et on définit sa position, qui sera l'ancienne position de notre fantôme. Maintenant, on peut charger notre dessin avec la routine pygame.image.load, en lui

passant en argument le nom du fichier (et le chemin si le fichier n'est pas au même endroit que le programme). Puis on récupère une référence (self.rect) vers le fantôme, ce qui règle la largeur et la hauteur du rectangle automatiquement, et on règle les positions x et y de ce rectangle aux positions passées à la routine.

La routine metAJour fait simplement une copie du fantôme, puis vérifie s'il sort de l'écran. Si c'est le cas, on le laisse où il était, sinon on change sa position de la valeur passée en argument.

Juste après l'instruction ecran.fill, placez le code de la page suivante (à droite).

Ici nous créons une instance de notre classe, appelée personnage. Puis on affiche le fantôme et on crée le rectangle fantôme vide que l'on remplit avec la couleur de fond. On rafraîchit la surface et on entre dans une boucle.

Tant que faireBoucle vaut 1, on boucle sur ce code. On utilise pygame.event.get() pour récupérer un caractère au clavier, puis on le compare à un type d'événement : si c'est QUIT, on sort du programme ; si c'est une frappe sur une touche, on le traite : on regarde quelle touche a été frappée (en utilisant les constantes définies par Pygame), puis on appelle la routine metAJour de notre classe. Notez que l'on envoie simplement une liste contenant le nombre de pixels sur les axes X et Y pour déplacer le fantôme. On le déplace de 10 pixels (+10 pour aller à droite ou en bas, - 10 pour aller en haut ou à gauche). Si la touche frappée est « q », on règle faireBoucle à 0 pour sortir de la boucle. Après tout cela, on affiche le rectangle vide à l'ancienne position, on affiche le fantôme à la nouvelle position, et pour finir, on rafraîchit, mais seulement les deux rectangles (le vide et celui qui contient le fantôme), ce qui gagne beaucoup de temps et évite des calculs inutiles.

```
class Fantome(pygame.sprite.Sprite):
    def __init__(self, position):

        pygame.sprite.Sprite.__init__(self)
        # sauve une copie du rectangle d'ecran
        self.ecran = pygame.display.get_surface().get_rect()
        # cree une variable pour stocker la position precedente du fantome
        self.ancienFantome = (0, 0, 0, 0)
        self.image = pygame.image.load('stick.png')
        self.rect = self.image.get_rect()
        self.rect.x = position[0]
        self.rect.y = position[1]
    def metAJour(self, valeur):
        # cree une copie du rectangle courant utilisee pour l'effacer
        self.ancienFantome = self.rect
        # deplace le rectangle de la valeur specifiee
        self.rect = self.rect.move(valeur)
        # verifie si on est sorti de l'ecran
        if self.rect.x < 0:
            self.rect.x = 0
        elif self.rect.x > (self.ecran.width - self.rect.width):
            self.rect.x = self.ecran.width - self.rect.width
        if self.rect.y < 0:
            self.rect.y = 0
        elif self.rect.y > (self.ecran.height - self.rect.height):
            self.rect.y = self.ecran.height - self.rect.height
```

(…)

Comme toujours, le code complet est disponible sur : www.thedesignatedgeek.com ou sur : http://fullcirclemagazine.pastebin.com/OrLRHKqY.

Pygame peut faire énormément de choses supplémentaires. Je vous suggère d'aller faire un tour sur leur site et de regarder la page des références (http://pygame.org/docs/ref/index.html). Vous pouvez aussi jeter un coup d'oeil aux jeux que d'autres gens ont déposés.

La prochaine fois, nous plongerons encore plus loin dans Pygame en créanr un jeu qui resurgit de mon passé... mon passé très LOINTAIN.

```
personnage = Fantome((ecran.get_rect().x, ecran.get_rect().y))
ecran.blit(personnage.image, personnage.rect)

# cree une surface de la taille de notre personnage
rectangleBlanc = pygame.Surface((personnage.rect.width,
personnage.rect.height))
rectangleBlanc.fill(couleurFond)

pygame.display.update()
faireBoucle = 1
while faireBoucle:
    for evenement in pygame.event.get():
        if evenement.type == pygame.QUIT:
            sys.exit()
        # verifie s'il y a un deplacement
        elif evenement.type == pygame.KEYDOWN:
            if evenement.key == pygame.K_LEFT:
                personnage.metAJour([-10, 0])
            elif evenement.key == pygame.K_UP:
                personnage.metAJour([0, -10])
            elif evenement.key == pygame.K_RIGHT:
                personnage.metAJour([10, 0])
            elif evenement.key == pygame.K_DOWN:
                personnage.metAJour([0, 10])
            elif evenement.key == pygame.K_q:
                faireBoucle = 0

    # efface l'ancienne position en y recopiant notre surface blanche
    ecran.blit(rectangleBlanc, personnage.ancienFantome)
    # dessine la nouvelle position
    ecran.blit(personnage.image, personnage.rect)
    # metAJour SEULEMENT les parties modifiees de l'ecran
    pygame.display.update([personnage.ancienFantome, personnage.rect])
```

TUTORIEL Programmer en Python - Partie 16

Greg Walters

Il y a quelque temps, j'ai promis à quelqu'un que je parlerais des différences entre Python 2.x et 3.x. Le mois dernier, j'ai écrit que nous continuerions notre programmation avec pygame, mais j'ai pensé que je devais tenir ma promesse, donc nous nous replongerons dans pygame la prochaine fois.

Il y a eu de nombreux changements dans Python 3.x. On trouve plein d'informations sur ces changements sur le web, et j'ai indiqué quelques liens à la fin de cet article. On trouve aussi pas mal de questions sur la manière de changer de version. Je vais me concentrer sur les changements qui affectent ce que nous avons appris jusqu'à maintenant.

C'est parti.

La commande PRINT

Comme je l'ai déjà écrit, l'un des problèmes les plus importants est la façon dont on utilise la commande print. Sous 2.x, on peut simplement utiliser :

```
print "Ceci est un test"
```

et c'est tout. Cependant, sous 3.x, si on essaie cela, on obtient le message d'erreur ci-dessous :

```
>>> print "Ceci est un test"
   File "<stdin>", line 1
     print "Ceci est un test"
                           ^
SyntaxError: invalid syntax
>>>
```

Pas terrible. Pour utiliser la commande print, on doit mettre ce que l'on veut afficher entre parenthèses, comme ceci :

```
print("Ceci est un test")
```

Ce n'est pas un gros changement, mais il faut y faire attention. Vous pouvez vous préparer à la migration en utilisant cette syntaxe sous Python 2.x.

Mise en forme et substitution de variable

La mise en forme et la substitution de variable ont également changé. Sous 2.x, nous avons utilisé une syntaxe comme dans l'exemple ci-dessous à gauche et, sous 3.1, on peut obtenir le bon résultat. Cependant, cela va changer car les fonctions de mise en forme '%s' et '%d' disparaissent. La nouvelle syntaxe est '{x}' et vous en verrez un exemple ci-dessous.

Il me semble que c'est plus facile à lire. Vous pouvez aussi faire des choses comme cela :

```
>>> print("Bonjour {0}. Je suis content que tu viennes sur {1}".format("Fred","MonSite.com"))

Bonjour Fred. Je suis content que tu viennes sur MonSite.com

>>>
```

Souvenez-vous que vous pouvez encore utiliser les fonctions '%s' et '%d', mais qu'elles vont disparaître.

Les nombres

Sous Python 2.x, si on faisait :

```
x = 5/2.0
```

x contenait 2.5. Mais si on faisait :

```
x = 5/2
```

x contenait 2 à cause de l'arrondi.

```
>>> mois = ['Jan','Fev','Mar','Avr','Mai','Juin','Juil','Aout','Sep','Oct','Nov','Dec']
>>> print "Vous avez choisi %s" % mois[3]
Vous avez choisi Avr
>>>
```

AVANT

```
>>> mois = ['Jan','Fev','Mar','Avr','Mai','Juin','Juil','Aout','Sep','Oct','Nov','Dec']
>>> print("Voux avez choisi {0}".format(mois[3]))
Vous avez choisi Avr
>>>
```

APRÈS

Sous 3.x, si on fait :

```
x = 5/2
```

on obtient 2.5. Pour arrondir la division, il faut faire :

```
x = 5//2
```

Les saisies

Il y a quelque temps, nous avons géré un système de menus en utilisant raw_input() pour demander une réponse de l'utilisateur de notre application. Cela ressemblait à ceci :

```
reponse = raw_input('Entrez
votre choix -> ')
```

Cela fonctionnaire sous 2.x. Mais sous 3.x, on obtient :

```
Traceback (most recent call
last):

File "<stdin>", line 1, in
<module>

NameError: name 'raw_input'
is not defined
```

Ce n'est pas un gros problème. La méthode raw_input() a été remplacée par input(). Changez simplement la ligne en :

```
reponse = input('Entrez
votre choix -> ')
```

et tout fonctionne correctement.

Non égalité

Sous 2.x, on pouvait tester une non-égalité avec « <> ». Mais ceci n'est pas autorisé avec 3.x. L'opérateur de test est maintenant « != ».

Convertir d'anciens programmes en Python 3.x

Un utilitaire est fourni avec Python 3.x pour aider à convertir les applications 2.x en code compatible avec la version 3.x. Cela ne fonctionne pas toujours, mais cela vous donnera un résultat approchant dans la plupart des cas. L'outil de conversion s'appelle (avec justesse) « 2to3 ». Prenons un programme très simple comme exemple. L'exemple ci-dessous vient de l'article « Programmer en Python, partie 3 ».

Quand on l'exécute sous 2.x, la sortie ressemble à ce que l'on voit ci-dessus à droite.

```
+====================================+
|Objet 1                       3.00 |
|Objet 2                      15.00 |
+------------------------------------+
| Total                       18.00 |
+====================================+
Fin du script.
```

Bien sûr, si on l'exécute en Python 3.x, cela ne fonctionne pas.

```
File "pprint1.py", line 18
print HautOuBas('=',40)
^
SyntaxError: invalid syntax
```

```
# pprint1.py
# Exemple de fonctions un peu utiles

def HautOuBas(caractere,largeur):
     # largeur est la largeur totale de la ligne retournee
     return '%s%s%s' % ('+',(caractere * (largeur-2)),'+')

def Fmt(val1,largGauche,val2,largDroite):
     # affiche 2 valeurs alignées avec des espaces
     # val1 sera affichee a gauche, val2 sera affichee a droite
     # largGauche=largeur de la partie de gauche, largDroite=largeur de la partie de droite
     partie2 = '%.2f' % val2
     return '%s%s%s%s' % ('| ',val1.ljust(largGauche-2,' '),part2.rjust(largDroite-2,'
'),' |')
# definit le prix de chaque objet
objet1 = 3.00
objet2 = 15.00
# maintenant on affiche tout...
print HautOuBas('=',40)
print Fmt('Objet 1',30,objet1,10)
print Fmt('Objet 2',30,objet2,10)
print HautOuBas('-',40)
print Fmt('Total',30,obbjet1+objet2,10)
print HautOuBas('=',40)
```

Essayons l'outil de conversion pour régler ce problème. Créons d'abord une sauvegarde de l'application à convertir. Je fais ça en créant une copie du fichier et en ajoutant « v3 » à la fin du nom de fichier :

```
cp pprint1.py pprint1v3.py
```

Il y a plusieurs façons d'exécuter l'application. Le plus simple est de laisser l'application vérifier notre code et nous dire où se situent les problèmes, ce qu'on voit ci-dessous à gauche.

Notez que le code source original n'est pas modifié. Il faut utiliser l'option « -w » pour signifier qu'on veut écrire les changements dans le fichier, ce qu'on peut voir ci-dessous à droite.

Vous remarquerez que la sortie est la même. Mais cette fois-ci, notre fichier source (visible sur la page suivante) est modifié en un fichier « compatible avec la version 3.x ».

Maintenant, le programme fonctionne correctement sous 3.x. Et, puisqu'il était simple, il tourne toujours avec la version 2.x.

Est-ce que je passe tout de suite à 3.x ?

La plupart des problèmes sont les mêmes dans tout changement d'un langage de programmation. Les changements de syntaxe sont nombreux avec chaque nouvelle version. Des raccourcis comme += ou -= apparaissent à l'improvisten et, en fait, nous facilitent la vie.

Quels sont les inconvénients à migrer à la version 3.x tout de suite ? Eh bien, il y en a quelques uns. La plupart des bibliothèques de modules que nous avons utlisées ne sont pas

```
> 2to3 pprint1v3.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored pprint1v3.py
--- pprint1v3.py (original)
+++ pprint1v3.py (refactored)
@@ -15,9 +15,9 @@
 objet1 = 3.00
 objet2 = 15.00
 # maintenant on affiche tout...
-print HautOuBas('=',40)
-print Fmt('Objet 1',30,objet1,10)
-print Fmt('Objet 2',30,objet2,10)
-print HautOuBas('-',40)
-print Fmt('Total',30,objet1+objet2,10)
-print HautOuBas('=',40)
+print(HautOuBas('=',40))
+print(Fmt('Objet 1',30,objet1,10))
+print(Fmt('Objet 2',30,objet2,10))
+print(HautOuBas('-',40))
+print(Fmt('Total',30,objet1+objet2,10))
+print(HautOuBas('=',40))
RefactoringTool: Files that need to be modified:
RefactoringTool: pprint1v3.py
```

```
> 2to3 -w pprint1v3.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored pprint1v3.py
--- pprint1v3.py (original)
+++ pprint1v3.py (refactored)
@@ -15,9 +15,9 @@
  objet1 = 3.00
  objet2 = 15.00
  # maintenant on affiche tout...
-print HautOuBas('=',40)
-print Fmt('Objet 1',30,objet1,10)
-print Fmt('Objet 2',30,objet2,10)
-print HautOuBas('-',40)
-print Fmt('Total',30,objet1+objet2,10)
-print HautOuBas('=',40)
+print(HautOuBas('=',40))
+print(Fmt('Objet 1',30,objet1,10))
+print(Fmt('Objet 2',30,objet2,10))
+print(HautOuBas('-',40))
+print(Fmt('Total',30,objet1+objet2,10))
+print(HautOuBas('=',40))
RefactoringTool: Files that were modified:
RefactoringTool: pprint1v3.py
```

disponibles en version 3.x pour le moment. Des choses comme Mutagen, que nous avons utilisé dans un article précédent, ne sont pas encore disponibles. Bien que ce soit une pierre d'achoppement, cela ne veut pas dire qu'il faut complètement abandonner Python 3.x.

Je vous suggère de commencer à coder en utilisant la syntaxe 3.x dès maintenant. Python 2.6 supporte presque tout ce dont vous pourriez avoir besoin d'écrire à la façon 3.x. Ainsi, vous serez prêt à passer à la version 3.x le jour où vous devrez le faire. Si vous pouvez vous contenter des bibliothèques de modules standards, allez-y et faites le changement. Par contre, si vous utilisez d'autres modules, il vous faudra attendre que la bibliothèque correspondante sorte en version 3.x. Cela viendra.

Voici quelques liens qui m'ont semblé utiles. Le premier est la page de manuel de 2to3. Le deuxième pointe vers un document de 4 pages d'astuces qui m'ont semblé une bonne référence. Le troisième est ce que je considère comme le meilleur livre pour utiliser Python (enfin jusqu'à ce que j'arrive à écrire le mien).

À la prochaine.

```
# pprint1.py
# Exemple de fonctions un peu utiles

def HautOuBas(caractere,largeur):
      # largeur est la largeur totale de la ligne retournee
      return '%s%s%s' % ('+',(caractere * (largeur-2)),'+')

def Fmt(val1,largGauche,val2,largDroite):
      # affiche 2 valeurs alignées avec des espaces
      # val1 sera affichee a gauche, val2 sera affichee a droite
      # largGauche=largeur de la partie de gauche, largDroite=largeur de la partie de droite
      partie2 = '%.2f' % val2
      return '%s%s%s%s' % ('| ',val1.ljust(largGauche-2,' '),part2.rjust(largDroite-2,'
'),' |')
# definit le prix de chaque objet
objet1 = 3.00
objet2 = 15.00
# maintenant on affiche tout...
print(HautOuBas('=',40))
print(Fmt('Objet 1',30,objet1,10))
print(Fmt('Objet 2',30,objet2,10))
print(HautOuBas('-',40))
print(Fmt('Total',30,obbjet1+objet2,10))
print(HautOuBas('=',40))
```

Liens

Manuel de 2to3 :
http://docs.python.org/library/2to3.html

Passer de Python 2 à Python 3 (4 pages d'astuces) :
http://ptgmedia.pearsoncmg.com/imprint_downloads/informit/promotions/python/python2python3.pdf

Plongez dans Python 3 :
http://diveintopython3.org/

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume trois

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

Il ne s'agit de rien d'autre qu'une reprise de la série Programmer en Python, parties 17 à 21, numéros 43 à 47 ; pas de chichi, juste les faits.

Gardez à l'esprit la date de publication ; les versions actuelles du maté-riel et des logiciels peuvent être différentes de celles illustrées. Il vous est recommandé de bien vérifier la version de votre matériel et des logiciels avant d'essayer d'émuler les tutoriels dans ces numéros spéciaux. Il se peut que vous ayez des logiciels plus récents ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

Site web :
http://www.fullcirclemagazine.org/

Forums :
http://ubuntuforums.org/forumdisplay.php?f=270

IRC : #fullcirclemagazine on chat.freenode.net

Équipe éditoriale
Rédacteur en chef : Ronnie Tucker (pseudo : RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia (pseudo : admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcaster : Robin Catling (pseudo : RobinCatling)
podcast@fullcirclemagazine.org
Dir. comm. : Robert Clipsham (pseudo : mrmonday) -
mrmonday@fullcirclemagazine.org

TUTORIEL

Programmer en Python - Partie 17

Greg Walters

En terminant l'article précédent de notre série, j'ai reçu un courriel au sujet d'une compétition de programmation. Nous n'avons pas le temps de traiter celle-ci, mais plusieurs sites proposent des compétitions de programmation au cours de l'année. L'information concernant cette compétition, si vous êtes intéressé, se trouve ici : http://www.freiesmagazin.de/third programming contest. Ceci m'a fait penser que nous n'avons pas encore parlé de programmation client/serveur. Nous allons donc nous y lancer avec cette idée derrière la tête et nous verrons bien où cela nous mène.

Alors, qu'est-ce qu'une application client/serveur ? Pour faire simple, à chaque fois que vous utilisez un programme (ou même une interface web) qui accède à des données d'une autre application ou d'un autre ordinateur, vous utilisez un système client/serveur. Examinons un exemple que nous avons déjà utilisé. Vous souvenez-vous de notre programme de livre de recettes ? C'était un exemple TRÈS simple (et pas très bon) d'une application client/serveur. La base SQLite était le serveur, l'application que nous avons décrite était le client. L'exemple suivant serait meilleur : imaginez qu'une base de données soit sur un ordinateur à un autre endroit de votre entreprise, à un autre étage ; elle contient des informations sur l'inventaire du magasin dans lequel vous travaillez. Vous utilisez une caisse à un des dix points de vente dans le magasin : chacune de ces caisses est un client et la base de données située ailleurs est le serveur.

Même si nous ne cherchons pas à créer un tel système ici, nous pouvons apprendre quelques-unes des bases.

La première chose à laquelle il faut réfléchir est l'emplacement de notre serveur. De nombreuses personnes n'ont qu'un seul ordinateur à la maison. Certaines en ont parfois 7 ou 8.

Pour utiliser un système client/serveur, nous devons nous connecter à la machine serveur depuis la machine cliente. On fait cela avec ce qu'on appelle un « pipe » ou un « socket » [Ndt : un connecteur]. Si vous avez déjà fabriqué un téléphone avec deux boîtes de conserves quand vous étiez enfant, vous avez une idée de ce dont je vais vous parler. Sinon, laissez-moi vous décrire une scène d'autrefois. D'abord il fallait demander à votre mère qu'elle vous garde deux boîtes de conserves de haricots ou autre. Puis il fallait les nettoyer soigneusement et les apporter au garage. Ensuite, en utilisant un petit clou et un marteau, vous perciez un petit trou sur le fond de chaque boîte. Puis avec 40 cm de ficelle (que vous avait donné votre maman chérie), il fallait passer la ficelle dans les deux boîtes et faire un gros nœud à chaque bout à l'intérieur des boîtes. Enfin, vous alliez chercher votre meilleur copain et, en tendant bien fort la corde, vous pouviez crier dans une des boîtes pendant que le copain plaçait l'autre près de son oreille. Les vibrations du fond de la boîte traversaient la corde tendue et faisaient vibrer le fond de l'autre boîte. Bien sûr, vous pouviez entendre sans la boîte, mais c'était hors de propos. C'était chouette. Le « socket » est à peu près la même chose. Le client a une connexion directe (pensez à la ficelle) au serveur. Si plusieurs clients sont connectés au serveur, chaque client aurait sa propre boîte de conserves et le pauvre serveur devrait avoir le même nombre de boîtes avec des ficelles bien tendues jusqu'à chaque client. L'essentiel ici est que chaque client ait sa propre ligne jusqu'au serveur.

Fabriquons un serveur et un client simples. Commençons par le serveur. En pseudo-code, voici ce qui se passe.

```
Créer un connecteur.
Récupérer le nom du serveur.
Choisir un port.
Relier le connecteur à
l'adresse et au
port.
Écouter s'il y a une
connexion.
Si c'est le cas :
accepter la connexion ;
afficher qu'on est connecté ;
fermer la connexion.
```

Vous pouvez voir le code pour le serveur en bas à gauche de la page suivante.

Ainsi, on crée le connecteur, on récupère le nom de la machine sur laquelle tourne le serveur, on relie le

connecteur au port et on commence à écouter. Quand on reçoit une demande de connexion, on l'accepte, on affiche que l'on vient d'établir une connexion, on envoie « Bonjour et au revoir » et on ferme le connecteur.

Maintenant, il nous faut un client pour que l'ensemble fonctionne (voir en bas à droite).

Le code est presque le même que pour le serveur, mais cette fois-ci on se connecte, on affiche ce qu'on a reçu et on ferme la connexion.

Les sorties du programme sont très prévisibles. Du côté du serveur, on obtient :

```
Mon nom est terre.
Je suis connecte a
('127.0.0.1', 45879).
```

et du côté du client, on obtient :

```
Bonjour et au revoir.
```

C'est donc plutôt simple. Maintenant faisons quelque chose de plus réaliste. Nous allons créer un serveur qui va vraiment faire quelque chose. Le code pour la version 2 du serveur est disponible ici http://fullcirclemagazine.pastebin.com/jZJvqPym

Décortiquons-le. Après les « import », on règle quelques variables. BUFSIZ contient la taille du tampon qui sera utilisé pour contenir l'information que l'on recevra du client. On règle aussi le port sur lequel on va écouter et une liste contenant le nom de l'hôte et le numéro du port. Ensuite on crée une classe nommée « ServCmd ». Dans la routine init, on crée un connecteur et on relie l'inter-

```
#!/usr/bin/env python
#server1.py
import socket
soc = socket.socket()
nom_hote = socket.gethostname()
print "Mon nom est ", nom_hote
port = 21000
soc.bind((nom_hote,port))
soc.listen(5)
while True:
    con,adresse = soc.accept()
    print "Je suis connecte a ",adresse
    con.send("Bonjour et au revoir")
    con.close(
```

```
#!/usr/bin/env python
# client2.py
from socket import *
from time import time
from time import sleep
import sys
BUFSIZE = 4096
class CmdLine:
    def __init__(self,host):
        self.HOST = host
        self.PORT = 29876
        self.ADDR = (self.HOST,self.PORT)
        self.sock = None
    def seConnecte(self):
        self.sock = socket( AF_INET,SOCK_STREAM)
        self.sock.connect(self.ADDR)
    def envoieCommande(self, cmd):
        self.sock.send(cmd)
    def recupResultats(self):
        data = self.sock.recv(BUFSIZE)
        print data
if __name__ == '__main__':
    conn = CmdLine('localhost')
    conn.seConnecte()
    conn.envoieCommande('ls -al')
    conn.recupResultats()
    conn.envoieCommande('AU REVOIR')
```

```
#!/usr/bin/python
# client1.py
#====================
import socket
soc = socket.socket()
nom_hote = socket.gethostname()
port = 21000
soc.connect((nom_hote, port))
print soc.recv(1024)
soc.close
```

face à ce connecteur. Dans la routine « run », on commence à écouter et on attend une commande du client.

Lorsqu'on reçoit une commande du client, on utilise la routine os.popen() qui, en résumé, crée une invite de commande et exécute la commande.

Passons au client (en haut à droite, page précédente), qui est sensiblement plus simple. On ne va expliquer ici que la commande « send », car vous avez maintenant assez de connaissances pour comprendre le reste par vous-mêmes. La ligne coo.envoieCommande() (ligne 31) envoie une simple requête « ls -al ». Voici à quoi ressemblent mes réponses, les vôtres seront quelque peu différentes.

Serveur :

```
python server2.py
Ecoute le client...
Connecte a ('127.0.0.1',
42198)
Recu la commande : ls -al
Recu la commande : AU REVOIR
Ecoute le client...
```

Client :

```
python client2a.py total 72
drwxr-xr-x 2 greg greg 4096
2010-11-08 05:49 .
drwxr-xr-x 5 greg greg 4096
2010-11-04 06:29 ..
-rw-r--r-- 1 greg greg 751
2010-11-08 05:31 client2a.py
-rw-r--r-- 1 greg greg 760
2010-11-08 05:28
client2a.py~
-rw-r--r-- 1 greg greg 737
2010-11-08 05:25 client2.py
-rw-r--r-- 1 greg greg 733
2010-11-08 04:37 client2.py~
-rw-r--r-- 1 greg greg 1595
2010-11-08 05:30 client2.pyc
-rw-r--r-- 1 greg greg 449
2010-11-07 07:38 ping2.py
-rw-r--r-- 1 greg greg 466
2010-11-07 10:01
python_client1.py
-rw-r--r-- 1 greg greg 466
2010-11-07 10:01
python_client1.py~
-rw-r--r-- 1 greg greg 691
2010-11-07 09:51
python_server1.py
-rw-r--r-- 1 greg greg 666
2010-11-06 06:57
python_server1.py~
-rw-r--r-- 1 greg greg 445
2010-11-04 06:29 re-test1.py
-rw-r--r-- 1 greg greg 1318
2010-11-08 05:49 server2a.py
-rw-r--r-- 1 greg greg 1302
2010-11-08 05:30
server2a.py~
-rw-r--r-- 1 greg greg 1268
2010-11-06 08:02 server2.py
-rw-r--r-- 1 greg greg 1445
2010-11-06 07:50 server2.py~
-rw-r--r-- 1 greg greg 2279
2010-11-08 05:30 server2.pyc
```

On peut aussi se connecter depuis une autre machine sans aucun changement, à la seule exception de la ligne 29 : conn = CmdLine('localhost') dans le programme client. Dans ce cas, remplacez « localhost » par l'adresse IP de la machine sur laquelle tourne le serveur. Chez moi, j'utilise la ligne suivante :

```
conn =
CmdLine('192.168.2.12')
```

Et voilà, maintenant on peut envoyer de l'information d'une machine (ou d'un terminal) à une autre.

La prochaine fois, nous rendrons notre application client/serveur plus robuste.

Idées et auteurs souhaités

Nous avons créé les pages du projet Full Circle et de son équipe sur Launchpad. L'idée étant que des personnes qui ne sont pas auteurs puissent aller sur la page du projet, cliquer sur « Answers » [Ndt : Réponses] en haut de la page et laisser leurs idées d'article, mais merci d'être précis dans vos idées ! Ne laissez pas seulement « article sur les serveurs », spécifiez s'il vous plaît ce que le serveur devrait faire !

Les lecteurs qui aimeraient écrire un article, mais qui ne savent pas à propos de quoi écrire, peuvent s'inscrire sur la page de l'équipe du Full Circle, puis s'attribuer une ou plusieurs idées d'articles et commencer à écrire ! Nous vous demandons expressément, si vous ne pouvez terminer l'article en quelques semaines (au plus un mois), de rouvrir la question pour laisser quelqu'un d'autre récupérer l'idée.

La page du projet, pour les idées :
https://launchpad.net/fullcircle
La page de l'équipe pour les auteurs :
https://launchpad.net/~fullcircle

TUTORIEL

Programmer en Python - Partie 18

Greg Walters

La dernière fois, nous avons créé un système client/serveur très simple. Cette fois-ci, nous allons l'améliorer un peu. Le serveur est un plateau de jeu Tic-Tac-Toe (ou jeu du morpion). La partie client sert à faire les saisies et l'affichage.

Nous allons repartir du code du serveur de la dernière fois et le modifier au fur et à mesure. Si vous n'avez pas gardé ce code, rendez-vous sur http://fullcirclemagazine.pastebin.com/jZJvqPym pour récupérer le code source de la dernière fois avant de continuer. Le premier changement se situe dans la routine __init__ où nous initialisons deux variables supplémentaires : self.joueur et self.plateau. Le plateau de jeu est simplement un tableau, autrement dit une liste de listes. On peut y accéder comme ci-contre, en haut de la 2e colonne (c'est plus visuel qu'une liste plate). La liste contiendra nos données. Il y a trois entrées possibles par cellule : « - » signifie que la cellule est vide, « X » signifie qu'elle est occupée par le joueur 1 et « O » qu'elle est occupée par le joueur 2. La grille ressemble à ceci si on la représente en deux dimensions :

```
[0][0] | [0][1] | [0][2]
[1][0] | [1][1] | [1][2]
[2][0] | [2][1] | [2][2]
```

Donc, en repartant du code du serveur de la dernière fois, dans la routine __init__, ajoutez les lignes suivantes :

```
# Les trois prochaines lignes
sont nouvelles

self.joueur = 1

self.plateau = [['-','-','-'],
['-','-','-'],['-','-','-']]

self.run()
```

Les routines run, ecoute et servCmd sont inchangées, donc nous allons nous concentrer sur les changements de la routine exeCommande.

Dans l'article précédent, le serveur attendait une commande du client, puis l'envoyait à la routine os.popen. Cette fois-ci, nous allons analyser la commande envoyée. Dans notre cas, nous avons trois commandes différentes qui peuvent arriver : « Debut », « Deplace » et « AU REVOIR ». Lorsqu'on reçoit la commande « Debut », le serveur doit initialiser le plateau de jeu en plaçant des « - » partout, puis envoyer un « Afficher le plateau » au client.

La commande « Deplace » est une commande composée, dans le sens où elle contient la commande, mais aussi la position où le joueur souhaite se déplacer. Par exemple, « Deplace A3 ». On analyse la commande pour obtenir trois morceaux : la commande « Deplace » elle-même, la ligne et la colonne. Finalement, la commande « AU REVOIR » remet simplement le plateau à zéro pour un nouveau jeu.

Ainsi, on reçoit la commande du client dans la routine exeCommande. On examine ensuite la commande pour vérifier ce qu'on doit faire. Dans la routine exeCommande, cherchez la cinquième ligne et, après la ligne qui contient « if self.boucleGestion: », enlevez le reste du code. Maintenant nous allons régler les commandes comme nous l'avons prévu. Voici le code pour la commande « Debut » :

```
if self.boucleGestion:
if cmd == 'Debut':
self.InitialisePlateau()
self.AffichePlateau(1)
```

Ensuite, nous allons regarder la partie « Deplace » de la routine (voir ci-dessous). Nous vérifions d'abord les sept premiers caractères de la commande passée pour voir s'il s'agit de « Deplace ». Si oui, on prend alors de reste de la chaîne en commençant à la position 8 (puisqu'on compte à partir de 0), et on l'assigne à la variable position. On vérifie ensuite si le premier caractère est « A » ou « B » ou « C ». Il représente la ligne que le client a envoyée. On récupère ensuite l'entier dans le caractère suivant et c'est le numéro de colonne :

```
if cmd[:7] == 'Deplace':
print "COMMANDE DEPLACE"
position = cmd[8:]
if position[0] == 'A':
    ligne = 0
elif position[0] == 'B':
    ligne = 1
elif position[0] == 'C':
    ligne = 2
else:
    self.cli.send('Position
invalide')
    return
col = int(position[1])-1
```

Puis on vérifie rapidement que le numéro de la ligne est dans la plage de valeurs autorisées :

```
if ligne < 0 or ligne > 2:

self.cli.send('Position invalide')

return
```

Enfin, on vérifie que la position est libre (« - ») et, si le joueur actuel est le joueur 1, on y place un « X », sinon un « O ». On appelle ensuite la routine AffichePlateau avec le paramètre 0 :

```
if self.plateau[ligne][col] == '-':

  if self.joueur == 1:

    self.plateau[ligne][col] = "X"

else:

  self.plateau[ligne][col] = "O"

self.AffichePlateau(0)
```

Ceci termine des changements apportés à la routine exeCommande. Il faut maintenant écrire la routine « Initialiser le plateau de jeu ». Elle se contente de régler chaque position à « - », puisque la logique de déplacement utilise cela pour vérifier qu'un emplacement est libre :

```
def InitialisePlateau(self):
self.plateau = [['-','-','-'],['-','-','-'],['-','-','-']]
```

La routine AffichePlateau (ci-dessous) affiche le plateau de jeu, appelle la routine verifGagnant et règle le numéro du joueur. On construit une grande chaîne de caractères à envoyer au client pour qu'il n'ait à appeler la routine qu'une fois par tour. Le paramètre premiereFois est là pour envoyer un joli plateau vide quand le client se connecte pour la première fois ou qu'il réinitialise le jeu (en bas) :

Ensuite, on vérifie si le paramètre premiereFois vaut 0 ou 1 (ci-dessous). S'il vaut 0, on vérifie si le joueur courant a gagné et, si oui, on ajoute le texte « Joueur X GAGNE ! » à la chaîne de sortie. Si le joueur courant n'a pas gagné, on ajoute le texte « Saisir le déplacement... » à la chaîne de sortie. Finalement, on envoie la chaîne au client avec la routine cli.send :

Pour finir, sur la page suivante, vous verrez la routine verifGagnant. On a déjà réglé le joueur à « X » ou « O », donc on commence à utiliser une boucle « for » simple. Si on trouvez un gagnant, la routine renvoie True. La variable « C » représente chaque colonne dans notre liste de listes.

Le Client

À nouveau, on commence par la routine simple de la fois précédente. Les changements commencent juste

```
if premiereFois == 0:
    if self.joueur == 1:
        ret = self.verifGagnant("X")
    else:
        ret = self.verifGagnant("O")
    if ret == True:
        if self.joueur == 1:
            outp += "Joueur 1 GAGNE !"
        else:
            outp += "Joueur 2 GAGNE !"
    else:
        if self.joueur == 1:
            self.joueur = 2
        else:
            self.joueur = 1
        outp += ('Saisir le deplacement du joueur %s' %
self.joueur)
    self.cli.send(outp)
```

```
def AffichePlateau(self,premiereFois):
# affiche la premiere ligne
outp = ('   1   2   3') + chr(13) + chr(10)
outp += (" A {0} | {1} | {2}".format(self.plateau[0][0],self.plateau[0][1],self.plateau[0][2])) + chr(13)+chr(10)
outp += (' -----------')+ chr(13)+chr(10)
outp += (" B {0} | {1} | {2}".format(self.plateau[1][0],self.plateau[1][1],self.plateau[1][2]))+ chr(13)+chr(10)
outp += (' -----------')+ chr(13)+chr(10)
outp += (" C {0} | {1} | {2}".format(self.plateau[2][0],self.plateau[2][1],self.plateau[2][2]))+ chr(13)+chr(10)
outp += (' -----------')+ chr(13)+chr(10)
```

On commence par vérifier si une ligne horizontale est gagnante :

```
def verifGagnant(self,joueur):
    # boucle sur les lignes et colonnes
    for c in range(0,3):
    # verifie si on a une ligne horizontale
        if self.plateau[c][0] == joueur and
self.plateau[c][1] == joueur and self.plateau[c][2] == joueur:
          print "*********\n\n%s gagne\n\n*********" % joueur
          joueurGagne = True
          return joueurGagne
```

Puis on vérifie si une colonne est gagnante :

```
        # verifie si on a une ligne verticale
        elif self.plateau[0][c] == joueur and
self.plateau[1][c] == joueur and self.plateau[2][c] == joueur:
          print "*********\n\n%s gagne\n\n*********" % joueur
          joueurGagne = True
          return joueurGagne
```

Maintenant on vérifie si une diagonale de gauche à droite gagne...

```
        # verifie si on a une diagonale (gauche a droite)
        elif self.plateau[0][0] == joueur and
self.plateau[1][1] == joueur and self.plateau[2][2] == joueur:
         print "*********\n\n%s gagne\n\n*********" % joueur
          joueurGagne = True
          return joueurGagne
```

Puis de droite à gauche...

```
       #verifie si on a une diagonale (droite a gauche)
        elif self.plateau[0][2] == joueur and
self.plateau[1][1] == joueur and self.plateau[2][0] == joueur:
          print "*********\n\n%s gagne\n\n*********" % joueur
          joueurGagne = True
          return joueurGagne
```

Finalement, si rien ne gagne, on renvoie faux

```
   else:
        joueurGagne = False
        return joueurGagne
```

après l'appel à conn.seConnecte. On envoie un « Debut », plusieurs « Deplace » et, enfin, une commande « AU REVOIR ». La chose la plus importante ici est qu'on doit envoyer une commande, puis obtenir une réponse avant d'envoyer une autre commande. Pensez à une conversation polie. Effectuez votre demande, attendez la réponse, puis effectuez une autre demande, attendez la réponse et ainsi de suite. Dans cet exemple, on utilise raw_input simplement pour qu'on puisse voir ce qui se passe :

```
if __name__ == '__main__':
conn = CmdLine('local-
host')
conn.seConnecte()
conn.envoieCommande('De-
but')
conn.recupResultats()
conn.envoieCommande('De-
place A3')
conn.recupResultats()
r = raw_input("Presser
Entree")
conn.envoieCommande('De-
place B2')
conn.recupResultats()
r = raw_input("Presser
Entree")
```

Continuez la suite de routines envoieCommande, recupResultats, raw_input avec les commandes suivantes (le code pour A3 et B2 est déjà écrit) : C1, A1, C3, B3, C2 puis terminez par une commande « AU REVOIR ».

Aller plus loin

Et maintenant, voici un « devoir » à faire à la maison. Dans la partie client, enlevez les mouvements codés en dur et utilisez raw_input() pour demander au joueur de saisir ses déplacements sous la forme « A3 » ou « B2 », puis ajoutez la commande « Deplace » devant et envoyez le tout au serveur.

La prochaine fois, nous modifierons le serveur pour qu'il tienne le rôle de l'autre joueur.

Les codes sources complets du client et du serveur sont ici : http://fullcirclemagazine.pastebin.com/5iNkC5Fr ou là, en anglais : http://the-designatedgeek.com

TUTORIEL Programmer en Python - Partie 19

Greg Walters

Cette fois-ci, nous allons travailler sur la fin de notre programme de Tic-Tac-Toe. Cependant, contrairement à mes autres articles, je ne fournirai pas le code : vous le ferez ! Je vous donnerai quand même les règles du jeu. Après 18 mois, vous avez les outils et les connaissances pour terminer ce projet. J'en suis persuadé.

Tout d'abord, examinons la logique du jeu de Tic-Tac-Toe. Nous verrons cela sous forme de pseudo-code. Regardons d'abord le plateau de jeu, qui se présente ainsi :

```
Coin   |   Côté    |  Coin
-------+--------+-------
 Côté  | Centre   | Côté
-------+--------+-------
Coin   |   Côté    |  Coin
```

Celui qui est « X » commence à jouer. Le meilleur premier mouvement est de prendre un des coins. N'importe lequel, ça n'a pas d'importance. Nous traiterons en premier les permutations lorsque « X » joue ; on les voit en haut à droite.

Le point de vue du joueur « O » est indiqué en bas à droite.

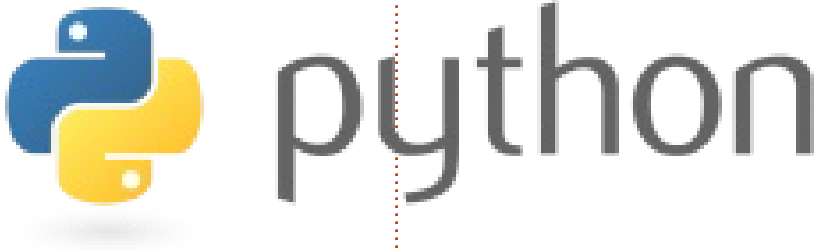
```
SI « O » prend un COIN ALORS
  # Scenario 1
     « X » devrait prendre un des coins restants. N'importe lequel.
     SI « O » bloque la victoire ALORS
     « X » prend le coin restant.
     Terminer en gagnant.
SINON
     Terminer en gagnant.
SINON SI « O » prend un CÔTÉ ALORS
   # Scénario 2
     « X » prend le CENTRE
SI « O » bloque la victoire ALORS
     « X » prend le coin qui n'est pas voisin d'un « O »
     Terminer en gagnant.
SINON
     Terminer en gagnant.
SINON
     # « O » a joué au CENTRE — Scénario 3
     « X » prend le coin opposé en diagonale au premier coup
     SI « O » joue dans un coin
     « X » joue dans le coin restant
     Terminer en gagnant.
SINON
     # le jeu sera nul — Scénario 4
     Bloquer la victoire de « O ».
     Bloquer tout victoire possible
```

Certaines possibilités de jeu sont indiquées à la page suivante.

Comme vous pouvez le voir, la logique est un peu compliquée, mais peut se ramener facilement à une suite d'instructions SI (notez que j'utilise ALORS, mais en Python on utilise plutôt «_:_»). Vous devriez arriver à modifier le code du mois dernier pour gérer tout cela, ou au moins écrire à partir de rien un programme simple de jeu de Tic-Tac-Toe de bureau.

```
SI « X » ne joue pas au centre
ALORS
     « O » prend le centre
     SI « X » a un coin ET
     un côté ALORS
  # Scénario 5
     « O » prend le coin
     Bloquer les victoires possibles pour
un nul.
     SINON
     # « X » a deux côtés
  # Scénario 6
     « O » prend le coin
     entouré par deux « X »
     SI « X » bloque la victoire ALORS
     « O » prend n'importe quelle case
     Bloquer et forcer un nul
          SINON
      Terminer en gagnant.
```

```
Scenario 1
X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | - | X    X | X | X
- | - | -    - | - | -    - | - | -    O | - | -    O | - | -    O | O | -    O | O | -
- | - | -    - | - | O    X | - | O    X | - | O    X | - | O    X | - | O    X | - | O

Scenario 2
X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | - | X    X | X | X
- | - | -    O | - | -    O | X | -    O | X | -    O | X | -    O | X | -    O | X | -
- | - | -    - | - | -    - | - | -    - | - | O    - | - | O    O | - | O    X | - | O

Scenario 3
X | - | -    X | - | -    X | - | -    X | - | X    X | O | X    X | O | X    X | O | X
- | - | -    - | O | -    - | O | -    - | O | -    - | O | -    - | O | -    - | O | X
- | - | -    - | - | -    - | - | X    O | - | X    O | - | X    O | - | X    O | - | X

Scenario 4
X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | O | X
- | - | -    - | O | -    - | O | O    X | O | O    X | O | O    X | O | O    X | O | O
- | - | -    - | - | -    - | - | X    - | - | X    O | - | X    O | - | X    O | - | X

Scenario 5
X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | X
- | - | -    - | O | -    - | O | X    - | O | X    X | O | X    X | O | X    X | O | X
- | - | -    - | - | -    - | - | -    - | - | O    - | - | O    O | - | O    O | - | O

Scenario 6
- | - | -    - | - | -    - | - | -    - | - | -    - | - | X    O | - | X    O | - | X
X | - | -    X | O | -    X | O | -    X | O | -    X | O | -    X | O | -    X | O | -
- | - | -    - | - | -    - | X | -    O | X | -    O | X | -    O | X | -    O | X | O
```



Idées et auteurs souhaités

Nous avons créé les pages du projet Full Circle et de son équipe sur Launchpad. L'idée étant que des personnes qui ne sont pas auteurs puissent aller sur la page du projet, cliquer sur « Answers » [Ndt : Réponses] en haut de la page et laisser leurs idées d'article, mais merci d'être précis dans vos idées ! Ne laissez pas seulement « article sur les serveurs », spécifiez s'il vous plaît ce que le serveur devrait faire !

Les lecteurs qui aimeraient écrire un article, mais qui ne savent pas à propos de quoi écrire, peuvent s'inscrire sur la page de l'équipe du Full Circle, puis s'attribuer une ou plusieurs idées d'articles et commencer à écrire ! Nous vous demandons expressément, si vous ne pouvez terminer l'article en quelques semaines (au plus un mois), de rouvrir la question pour laisser quelqu'un d'autre récupérer l'idée.

La page du projet, pour les idées :
https://launchpad.net/fullcircle
La page de l'équipe pour les auteurs :
https://launchpad.net/~fullcircle

TUTORIEL

Greg Walters

Programmer en Python - Partie 20

Nous voici de retour. Cette fois-ci nous allons refaire de la programmation graphique, mais en utilisant la bibliothèque pyGTK. Nous ne travaillerons pas avec un éditeur d'interfaces pour le moment, mais seulement avec la bibliothèque.

Utilisez Synaptic pour installer python-gtk2, python-gtk2-tutorial et python-gtk2-doc.

Et maintenant commençons tout de suite à écrire notre premier programme avec pyGTK ; voyez en haut à droite.

Nous allons travailler pendant un certain temps sur ce simple bout de code. Vous voyez une nouvelle instruction à la ligne 3 : « pygtk.require ('2.0') » signifie que l'application ne fonctionnera pas si le module pygtk n'est pas au moins en version 2.0. Dans la routine __init__, nous assignons une fenêtre à la variable self.fenetre (ligne 8), puis nous l'affichons (ligne 9). Souvenez-vous que la routine __init__ est appelée dès qu'on instancie la classe (ligne 13). Sauvegardez le code sous le nom « simple1.py ».

```
# simple.py
import pygtk
pygtk.require('2.0')
import gtk

class Simple:
     def __init__(self):
          self.fenetre = gtk.Window(gtk.WINDOW_TOPLEVEL)
          self.fenetre.show()
     def main(self):
          gtk.main()
if __name__ == "__main__":
     simple = Simple()
     simple.main()
```

Exécutez-le dans un terminal : vous verrez une simple fenêtre s'afficher quelque part sur votre bureau ; sur le mien, elle s'affiche dans le coin supérieur gauche du bureau. Pour terminer le programme, vous devrez utiliser Ctrl-C dans le terminal. Pourquoi ? Nous n'avons pas ajouté le code pour détruire et donc terminer l'application. C'est ce que nous allons faire maintenant. Ajoutez la ligne suivante avant self.fenetre.show() :

```
self.fenetre.connect("delete_event", self.evenement_supprimer)
```

Puis ajoutez la routine suivante après l'appel à gtk.main() :

```
def evenement_supprimer(self, widget, event, data=None):
    gtk.main_quit()
    return False
```

Sauvegardez votre appli sous le nom « simple2.py » et exécutez-la à nouveau dans un terminal. Désormais, lorsque vous cliquez sur le « X » de la barre de titre, l'application se terminera. Que se passe-t-il vraiment ici ? La première ligne que nous avons ajoutée (self.fenetre.connect...) relie l'événement de fermeture de fenêtre à une routine, en l'occurrence self.evenement_supprimer. En retournant « False » au système, cela détruit du même coup la fenêtre actuelle dans la mémoire système.

Je ne sais pas pour vous, mais moi je préfère que mes applications s'ouvrent au centre de l'écran, plutôt que n'importe où aléatoirement, ou dans un coin (où elles risquent d'être masquées par autre chose). Modifions donc le code : il suffit d'ajouter la ligne suivante avant la ligne self.fenetre.connect dans la fonction __init__ :

```
self.fenetre.set_position(gtk.WIN_POS_CENTER)
```

Comme vous pouvez le deviner, ceci règle la position de la fenêtre au centre de l'écran. Sauvegardez l'appli sous le nom « simple3.py » et exécutez-la.

C'est plus joli, mais il n'y a pas grand-chose à voir. Allez, essayons d'ajouter un composant. Si vous vous souvenez du temps où nous avons travaillé avec Boa Constructor, les composants sont simplement des contrôles prédéfinis que l'on peut ajouter à la fenêtre pour faire des choses. L'un des contrôles les plus simples à ajouter est un bouton. Ajoutons le code suivant juste après la ligne self.fenetre.connect dans la routine __init__ du programme précédent :

```
self.button = gtk.Button
("Ferme-moi")

self.button.connect("cli-
cked",self.clicBouton1,None)

self.fenetreadd(self.button)

self.bouton.show()
```

La première ligne définit le bouton et le texte sur sa surface. La ligne suivante permet de le connecter à un événement de clic. La troisième ligne ajoute le bouton à la fenêtre et la quatrième affiche le bouton sur la surface de la fenêtre. En regardant la ligne self.bouton.connect, vous pouvez voir qu'il y a trois arguments. Le premier est l'événement auquel on veut se connecter, le deuxième est la routine qui sera appelée lorsque l'événement surviendra, dans ce cas « self.clicBouton1 » et la troisième est l'argument (s'il existe) qui sera passé à la routine précédemment indiquée.

Ensuite, nous devons créer la routine self.clicBouton1. Ajoutez ceci après la routine self.evenement_supprimer :

```
def clicBouton1(self,widget,da-
ta=None):
print "Clic bouton 1"
gtk.main_quit()
```

Comme vous pouvez le voir, la routine ne fait pas grand-chose. Elle affiche « Clic bouton 1 » dans le terminal, puis appelle la routine gtk.main_quit(), ce qui fermera la fenêtre et quittera l'application (comme si vous aviez cliqué sur le « X » dans la barre de titre). Sauvegardez à nouveau tout cela sous le nom « simple4.py » et exécutez-le dans un terminal. Vous verrez notre fenêtre centrée avec un bouton affichant « Ferme-moi ». Cliquez dessus et l'application se fermera, comme prévu. Notez cependant que la fenêtre est un peu plus petite que dans l'application « simple3 ». Vous pouvez redimensionner l'application, mais le bouton s'agrandit avec. Pourquoi ? Eh bien, nous avons simplement placé un bouton dans la fenêtre et la fenêtre s'est dimensionnée pour s'adapter au contrôle.

Nous avons en quelque sorte violé les règles de la programmation graphique en plaçant le bouton directement dans la fenêtre, sans utiliser de conteneur. Souvenez-vous lorsque nous avons utilisé Boa Constructor dans le premier article sur la programmation graphique, nous avions utilisé des boîtes de dimensionnement (des conteneurs) pour contenir nos contrôles. Nous devrions faire pareil, même si nous n'avons qu'un seul contrôle. Pour notre exemple suivant, nous ajouterons une Hbox (une boîte horizontale) pour contenir le bouton, ainsi qu'un deuxième bouton. Pour un conteneur vertical, nous utiliserions Vbox.

Pour commencer, utilisez « simple4.py » comme code de base. Effacez tout ce qui se trouve entre les lignes self.fenetre.connect(...) et self.fenetre.show(). C'est là que nous allons ajouter nos nouvelles lignes. Voici le code pour la Hbox et le premier bouton :

```
self.box1 = gtk.HBox(False,0)

self.fenetre.add(self.box1)

self.bouton = gtk.Button("Bou-
ton 1")

self.bouton.connect("cli-
cked",self.clicBouton1,None)

self.box1.pack_start(self.bou-
ton,True,True,0)

self.bouton.show()
```

Tout d'abord, nous ajoutons une Hbox appelée self.box1 ; les paramètres passés à Hbox sont homogènes (vrai ou faux) et une valeur d'espacement :

```
Box = gtk.HBox(homoge-
neous=False, spacing=0)
```

Le paramètre « homogeneous » indique si chaque widget contenu dans la boîte a la même taille (largeur dans le cas d'une Hbox et hauteur pour une Vbox). Dans ce cas, nous choisissons « false » (faux) et une valeur d'espacement de 0. Ensuite, nous ajoutons la boîte dans la fenêtre. Puis nous créons le bouton comme précédemment et connectons l'événement de clic à notre routine.

Maintenant nous arrivons à une nouvelle commande. La commande self.box1.pack_start sert à ajouter le bouton au conteneur (Hbox). Nous utilisons cette commande au lieu de la commande self.fenetre.add pour les widgets que nous voulons placer dans le conteneur. La commande (comme ci-dessus) est :

```
box.pack_start(widget,expand=
True, fill=True, padding=0)
```

La commande pack_start prend les paramètres suivants : en premier le widget, puis expand (True ou False) et une valeur de « padding ». L'espacement dans un conteneur représente l'espace entre les widgets ; le padding concerne les côtés droit et gauche de chaque widget. L'argument expand permet de choisir si les widgets dans la boîte rempliront tout l'espace de la boîte (True), ou bien si la boîte se rétrécira pour s'adapter

aux widgets (False). L'argument fill (remplir) n'a d'effet que si l'argument expand est à True. Finalement on affiche le bouton. Voici maintenant le code pour le deuxième bouton :

```
self.bouton2 = gtk.Button("Bou-
ton 2")

self.bouton2.connect("cli-
cked",self.clicBouton2,None)

self.box1.pack_start(self.bou-
ton2,True,True,0)

self.bouton2.show()

self.box1.show()
```

Remarquez que le code est à peu près le même que celui pour le premier bouton. La dernière ligne de code affiche la boîte.

À présent il faut ajouter la routine self.clicBouton2. Après la routine self.clicBouton1, ajoutez le code suivant :

```
def btn2Clicked(self,widget,da-
ta=None):

    print "Clic bouton 2"
```

et commentez la ligne suivante dans la routine self.clicBouton1 :

```
gtk.main_quit()
```

Nous voulons que les deux boutons affichent leur message « Clic bouton X » sans fermer la fenêtre.

Sauvegardez cela dans « simple4a.py » et exécutez-le dans un terrminal. Vous verrez une fenêtre centrée avec deux boutons (collés aux bords de la fenêtre) affichant « Bouton 1 » et « Bouton 2 ». Cliquez dessus et remarquez qu'ils répondent correctement aux clics comme nous l'avons prévu.

Maintenant, avant de refermer la fenêtre, redimensionnez-la (tirez sur le coin inférieur droit) et remarquez que les boutons grandissent et rétrécissent de façon égale lors du redimensionnement de la fenêtre. Pour comprendre le paramètre expand, modifiez le code de la fonction self.box1.pack_start en remplaçant True par False dans les deux lignes.

Exécutez à nouveau le programme et regardez ce qui se passe : cette fois-ci, la fenêtre semble identique au départ, mais lorsque vous la redimensionnez, les boutons restent de la même largeur, et il y a de l'espace vide à droite lorsque la fenêtre grandit. Remettez ensuite le paramètre expand à True et réglez le paramètre fill à False. Exécutez à nouveau le programme et maintenant les boutons restent de la même largeur et de l'espace apparaît à droite et à gauche des boutons lorsque la fenêtre grandit. Souvenez-vous que le paramètre fill ne sert à rien si le paramètre expand est à False.

Une autre façon de placer les widgets est d'utiliser un tableau. Souvent, si tout ce que vous placez peut tenir facilement dans une structure quadrillée, le meilleur choix (et le plus facile) est d'utiliser un tableau. Un tableau est comme une grille de tableur avec des lignes et des colonnes contenant des widgets. Chaque widget peut occuper une ou plusieurs cellules suivant ce que vous souhaitez en faire. Peut-être que le dessin suivant vous aidera à visualiser les possibilités. Voici une grille 2 sur 2 :

```
0             1             2
0+-----------+-----------+
 |           |           |
1+-----------+-----------+
 |           |           |
2+-----------+-----------+
```

Nous placerons deux boutons sur la première ligne, un dans chaque colonne. Sur la deuxième ligne, nous placerons un bouton qui s'étendra sur les deux colonnes. Comme ceci :

```
0             1             2
0+-----------+-----------+
 | Bouton 1  | Bouton 2  |
1+-----------+-----------+
 |         Bouton 3      |
2+-----------+-----------+
```

Pour dessiner un tableau, on crée un objet de type table et on l'ajoute à la fenêtre. L'instruction pour créer un tableau est :

```
Table = gtk.Table(lignes=1,co-
lonnes=1,homogeneous=True)
```

Si le drapeau « homogeneous » vaut True, la taille des cases du tableau sera calculée en fonction du plus haut widget dans la même ligne et du plus large dans la même colonne. On crée ensuite un widget (comme le bouton ci-dessus) puis on le place dans la bonne case du tableau. Pour le placer, on fait comme ceci :

```
table.attach(widget,gauche,
droite,haut,bas,xoptions=EX-
PAND|FILL,yoptions=EXPAND|
FILL, xpadding=0,ypadding=0)
```

Les seuls paramètres obligatoires sont les cinq premiers. Ainsi, pour placer un bouton dans le tableau en ligne 0 et colonne 0, on peut utiliser l'instruction suivante :

```
table.attach(buttonx,0,1,0,1)
```

Si on voulait le placer en ligne 0 et colonne 1 (souvenez-vous que la numérotation part de 0) comme le bouton ci-dessus, on écrirait :

```
table.attach(buttonx,1,2,0,1)
```

J'espère que c'est aussi clair que de la boue maintenant pour vous. Commençons avec notre code et vous comprendrez mieux. Tout d'abord les parties communes :

```
# table1.py
import pygtk
pygtk.require('2.0')
import gtk
class Table:
    def __init__(self):

        self.fenetre = gtk.
Window(gtk.WINDOW_TOPLEVEL)

        self.fenetre.set_posi-
tion(gtk.WIN_POS_CENTER)

        self.fenetre.set_-
title("Test Table 1")

        self.fenetre.set_bor-
der_width(20)

        self.fenetre.set_size_-
request(250, 100)

        self.fenetre.
connect("delete_event",

self.evenement_supprimer)
```

Il y a plusieurs choses nouvelles ici que nous allons étudier avant d'aller plus loin. La ligne 9 règle le titre de la fenêtre à « Test Table 1 ». On utilise l'instruction set_border_width pour dessiner une bordure de 20 pixels autour de la fenêtre avant d'y placer les widgets. Enfin, on force la taille de la fenêtre à 250x100 pixels en utilisant la fonction set_size_request. Cela vous paraît clair jusqu'ici ? Maintenant, créons le tableau et ajoutons-le à la fenêtre :

```
table = gtk.Table(2, 2, True)
# cree une grille 2x2

self.fenetre.add(table)
```

Ensuite, on crée un premier bouton, on règle l'événement pour le clic, on le place dans le tableau et on l'affiche :

```
bouton1 = gtk.Button("Bouton
1")

bouton1.connect("clicked",
self.gererClic,"bouton 1")

table.attach(bouton1,0,1,0,1)

bouton1.show()
```

Maintenant, le deuxième bouton :

```
bouton2 = gtk.Button("Bouton
2")
bouton2.connect("clicked",
self.gererClic,"bouton 2")

table.attach(bouton2,1,2,0,1)

bouton2.show()
```

C'est presque exactement pareil que pour le bouton 1, mais remarquez le changement dans l'appel à table.attach. Remarquez aussi que la routine que nous utiliserons lors des clics s'appelle self.callback, et est la même pour les deux boutons. Assez pour le moment, vous comprendrez mieux ce que nous faisons un peu plus tard.

Maintenant, le troisième bouton. Ce sera le bouton pour quitter :

```
bouton3 = gtk.Button("Quit-
ter")

bouton3.connect("clicked",
self.Quitter,"bouton 3")

table.attach(bouton3,0,2,1,2)

bouton3.show()
```

Pour finir, on affiche le tableau et la fenêtre. Voici aussi la routine principale et la routine de suppression que nous avons utilisées précédemment :

```
    table.show()

    self.fenetre.show()

def main(self):
    gtk.main()

def evenement_supprimer(self,
widget, event, data=None):

    gtk.main_quit()

    return False
```

Maintenant, voici la partie rigolote. Pour les boutons 1 et 2, on règle la routine de gestion d'événement à self.gereClic. Voici le code pour faire cela :

```
def gererClic(self,widget,da-
ta=None):

    print "clic sur %s" % data
```

Lorsque l'utilisateur clique sur le bouton, l'événement de clic est déclenché et les données fournies lorsqu'on a réglé la connexion à l'événement sont envoyées. Pour le bouton 1, les données envoyées sont « bouton 1 » et « bouton 2 » pour le bouton 2. Tout ce que nous avons à faire ici est d'afficher « clic sur X » dans le terminal. Je suis certain que vous voyez l'intérêt de cet outil lorsqu'on le combine avec une routine bien structurée du type SI | SINON SI| SINON.

Pour terminer, on doit définir la routine Quitter, utilisée lorsqu'on

clique sur le bouton Quitter :

```
def Quitter(self, widget,
event, data=None):

    print "Le bouton Quitter
a ete utilise"

    gtk.main_quit()
```

Et maintenant, le code principal final :

```
if __name__ == "__main__":

table = Table()

table.main()
```

Combinez tout ce code dans une application appelée « table1.py » et exécutez-la dans un terminal.

Pour récapituler, lorsqu'on veut utiliser pyGTK pour créer une application graphique, les étapes sont :

- Créer la fenêtre.
- Créer des Hbox, Vbox ou tableaux pour contenir vos widgets.
- Utiliser pack ou attach pour placer les widgets (selon que vous utilisez des « box » ou des tableaux).
- Afficher les widgets.
- Afficher les « box » ou le tableau.
- Afficher la fenêtre.

Maintenant, nous possédons de nombreux outils et connaissances pour aller plus loin. Tout le code est sur Pastebin : http://fullcirclemagazine.pastebin.com/XxaS0mvJ.

À la prochaine.

EXTRA ! EXTRA !
LISEZ CECI

LE SERVEUR PARFAIT ÉDITION SPECIALE

Il s'agit d'une édition spéciale du Full Circle qui est une réédition directe des articles Le Serveur parfait qui ont déjà été publiés dans le FCM n° 31 à 34.

http://fullcirclemagazine.org/special-edition-1-the-perfect-server/

Des éditions spéciales du magazine Full Circle sont sorties dans un monde sans méfiance*

PYTHON ÉDITION SPECIALE n° 1

Il s'agit d'une reprise de Programmer en Python, parties 1 à 8 par Greg Walters.

http://fullcirclemagazine.org/python-special-edition-1/

* Ni Full Circle magazine, ni ses concepteurs ne s'excusent pour l'hystérie éventuellement causée par la sortie de ces publications.

TUTORIEL

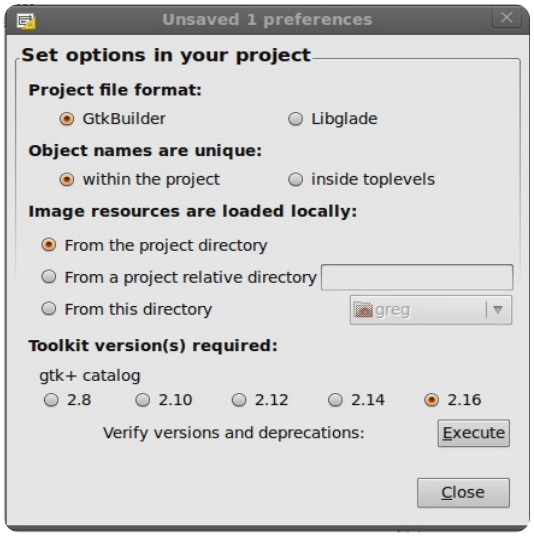
Programmer en Python - Partie 21

Greg Walters

Si vous avez suivi mes articles depuis un certain temps, vous vous souvenez peut-être des parties 5 et 6. Nous avions parlé de l'utilisation de Boa Constructor pour créer l'apparence de notre application graphique. Eh bien, cette fois-ci nous allons utiliser Glade Designer. C'est différent, mais similaire. Vous pouvez l'installer depuis la Logithèque Ubuntu : cherchez glade et installez « GTK+ 2 User Interface Builder ».

Juste pour vous annoncer le programme, nous allons créer une application pour laquelle nous allons avoir besoin de plusieurs articles pour arriver au bout. Le but final est de construire un programme pour créer des listes de lecture pour nos fichiers MP3 et autres fichiers de médias. Ce chapitre du tutoriel sera orienté vers la partie interface graphique. La prochaine fois, nous parlerons du code qui permet de rassembler les différentes parties de l'interface.

Commençons à concevoir notre application. Lorsque vous démarrez Glade Designer, vous aurez une fenêtre de préférences (haut de la deuxième colonne).

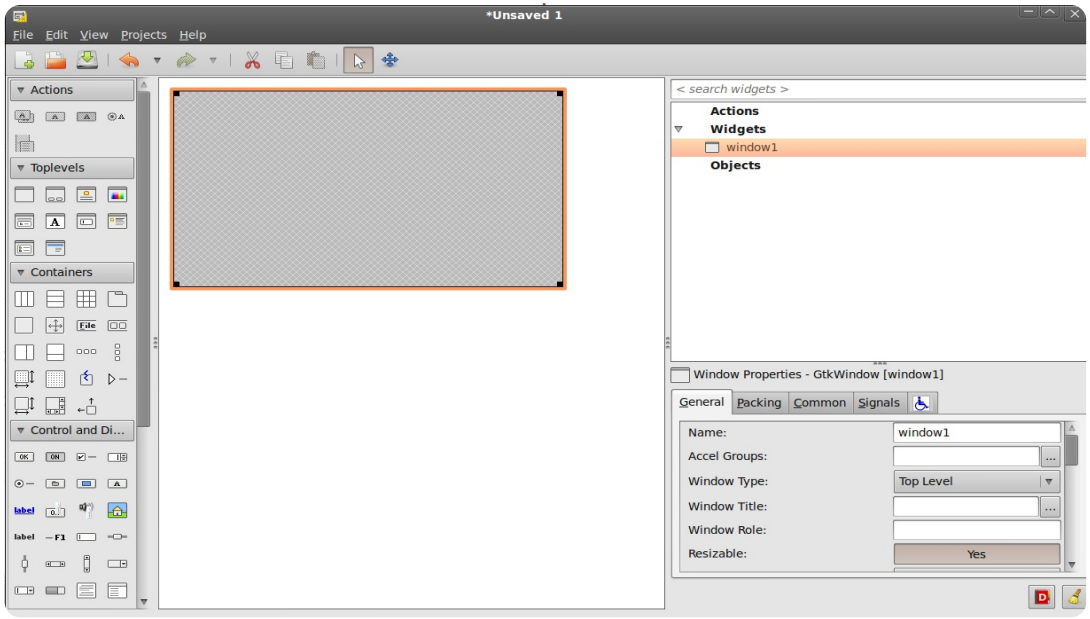


Choisissez Libglade et « à l'intérieur des niveaux supérieurs » puis cliquez sur Fermer. Cela nous amène à la fenêtre principale.

Jetons un coup d'œil à la fenêtre principale (à droite). Sur la gauche se trouve la boîte à outils, au milieu la zone de conception et, à droite, les attributs et les zones de hiérarchie.

Dans la boîte à outils, cherchez le groupe appelé « Niveaux supérieurs » et cliquez sur le premier outil (si vous le survolez avec la souris, il devrait afficher « Fenêtre »). Cela nous donnera notre « Canvas », une fenêtre vide avec laquelle nous allons travailler.

Remarquez que dans la zone de hiérarchie vous voyez window1 dans la section Widgets. Maintenant, descendez dans la section des attributs, modifiez le nom window1 en FenetrePrincipale et réglez le titre de fenêtre à « Créateur de liste de lecture v1.0 ». Sauvegardez votre travail sous le nom CreateurListeDeLecture.glade. Avant de continuer, cherchez dans la section des attributs de l'onglet Général la liste déroulante « Position de la fenêtre » et réglez-la sur « Centre ». Cochez la case « Largeur par défaut » et indiquez 650. Faites la même chose pour la case « Hauteur par défaut » et indiquez 350. Puis cliquez sur l'onglet « Commun » et descendez jusqu'à l'attribut « Visible ». ASSUREZ-VOUS DE LE RÉGLER SUR « OUI », sinon votre fenêtre ne s'affichera pas. Enfin, allez sur l'onglet Signaux, descendez jusqu'à la section GtkObject et ouvrez-la ; dans « destroy », cliquez sur la liste déroulante de la colonne Gestionnaire et choisissez on_FenetrePrincipale_destroy. Ceci crée un événement qui sera appelé lorsque l'utilisateur fermera notre fenêtre en cliquant sur le « X » dans la barre de titre. Un mot d'avertissement : après avoir réglé l'événement destroy, cliquez quelque part au-dessus ou en



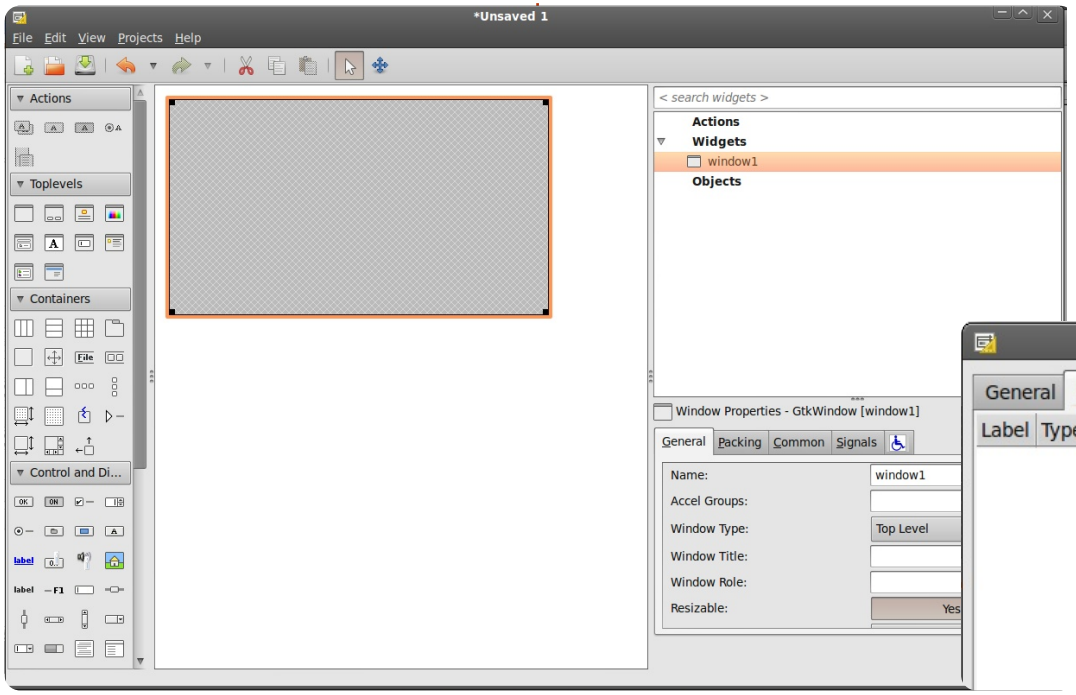
dessous pour valider le changement. Ceci semble être un bogue de Glade Designer. À nouveau, sauvegardez votre projet.

Comme la dernière fois que nous avons conçu une interface graphique, nous devons placer nos widgets dans des « vbox » et des « hbox ». C'est la chose la plus difficile à retenir lorsqu'on fait de la programmation graphique. Nous allons ajouter dans la fenêtre une boîte verticale pour contenir nos widgets ; choisissez « boîte verticale » dans la section Conteneurs de la boîte à outils (deuxième icône de la première ligne), et cliquez sur notre fenêtre vide dans la partie conception. Vous verrez apparaître une boîte de dialogue vous demandant combien d'éléments vous souhaitez. Par défaut, c'est 3, mais nous en voulons 5 : en partant du haut, nous placerons une barre d'outils, une zone pour une vue arborescente, deux zones horizontales pour des étiquettes, des boutons et des zones de saisie de texte, et une barre d'état.

Maintenant nous pouvons commencer à ajouter nos widgets. Tout d'abord, ajoutez une « Barre d'outils » depuis la boîte à outils. Chez moi, c'est la quatrième icône de la deuxième ligne des Conteneurs. Cliquez sur la rangée la plus haute de la vbox ; cette rangée va se rétrécir et presque disparaître, mais ne vous inquiétez pas, nous la récupérerons dans quelques minutes.

Ensuite nous allons ajouter une fenêtre avec ascenseur dans la rangée suivante pour y placer une vue arborescente ; ceci nous permettra de nous déplacer dans la vue. Cherchez l'icône de la fenêtre avec défilement dans la section des Conteneurs de la boîte à outils (chez moi c'est la deuxième icône de la cinquième rangée), et cliquez sur la deuxième rangée de la vbox. Ensuite, nous allons ajouter deux boîtes horizontales dans les deux rangées suivantes ; chacune devra avoir trois éléments. Enfin, ajoutez une barre d'état dans la rangée du bas : vous la trouverez dans la section « Contrôle et affichage » de la boîte à outils, près du bas. Maintenant, votre Glade Designer devrait ressembler à l'image ci-dessous.

Pour terminer, ajoutez à la fenêtre avec défilement une « Vue arborescente » que vous trouverez dans la section « Contrôle et affichage » de la boîte à outils. Vous obtiendrez une boîte de dialogue demandant quel modèle vous voulez utiliser. Cliquez simplement sur le bouton OK pour l'instant, nous nous occuperons de cela plus tard.



Maintenant nous allons nous intéresser quelques instants à la fenêtre avec défilement. Cliquez dessus dans la zone de hiérarchie. Défilez dans l'onglet « Général » jusqu'à trouver « Politique d'affichage des barres de défilement horizontales » ; modifiez cela pour afficher « Toujours » et faites la même chose pour la Politique d'affichage des barres de défilement verticales. Sauvegardez à nouveau.

Bon, maintenant intéressons-nous à la barre d'outils. Cette zone sera située tout en haut de notre application, juste en dessous de la barre de titre. Elle contiendra divers boutons, qui feront la plus grande partie du travail. Nous utiliserons onze boutons dans la barre d'outils, qui sont, de gauche à droite : ajouter, supprimer, effacer la liste, un séparateur, déplacer tout en haut, monter, descendre, déplacer tout en bas, un autre séparateur, à propos et quitter.

Dans la zone de hiérarchie, cliquez sur « barre d'outils 1 ». Elle devrait passer en surbrillance. Tout en haut de Glade Designer, vous verrez quelque chose qui ressemble à un crayon : cliquez dessus. Cela appelle l'éditeur de barres d'outils. Cliquez sur l'onglet « Hiérarchie » et vous verrez quelque chose comme l'image de la page précédente, en bas à droite.

Nous ajouterons tous les boutons de notre barre d'outils à partir de là. Les étapes seront :

- cliquer sur le bouton ajouter ;
- modifier le nom du bouton ;
- modifier l'étiquette du bouton ;
- choisir l'image.

Nous répéterons cela pour nos onze widgets. Allez, cliquez sur « Ajouter » puis dans la boîte « Nom » et saisissez « boBtnAjouter ». Défilez jusqu'à la partie « Modifier l'étiquette » et saisissez « Ajouter » dans la zone « étiquette » ; puis un peu plus bas vous trouverez « Modifier l'image » et dans la partie « ID prédéfini », utilisez la liste déroulante pour choisir « Ajouter ». Voilà pour notre bouton Ajouter. Nous l'avons nommé « boBtnAjouter » pour pouvoir y faire référence dans le code plus tard. « boBtn » est un raccourci pour « Bouton de barre d'outils ». Ainsi dans notre code il sera facile à repérer et est auto-documenté.

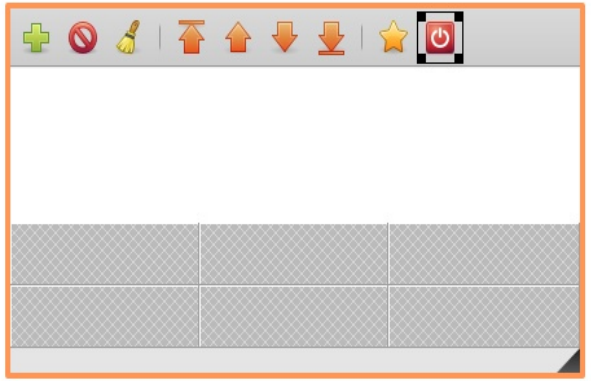
Maintenant il faut ajouter les autres widgets à notre barre d'outils. Ajoutez un autre bouton pour « Supprimer ». Celui-ci s'appellera (comme vous l'aurez deviné) « boBtnSupprimer ». À nouveau, réglez l'étiquette et l'icône. Puis ajoutez un autre bouton en le nommant « boBtnEffacer » et utilisez l'icône « Effacer ». Maintenant nous voulons placer un séparateur ; cliquez donc sur « Ajouter », nommez-le « Sep1 » et choisissez « Séparateur » dans la liste déroulante des types.

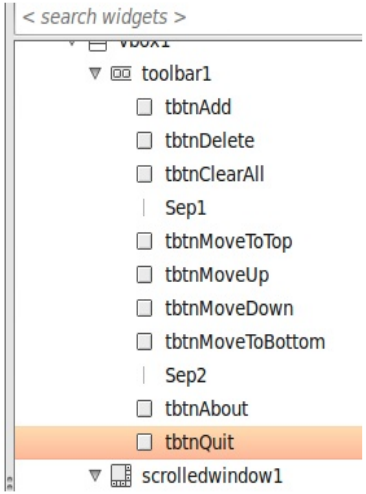
Ajoutez le reste des widgets en les nommant « boBtnHaut », « boBtnMonter », « boBtnDescendre », « boBtnBas », « Sep2 », « boBtnAPropos » et « boBtnQuitter ». Je suis sûr que vous trouverez les icônes correctes. Une fois cela terminé, vous pouvez quitter la fenêtre de hiérarchie et sauvegarder votre travail. Vous devriez avoir quelque chose qui ressemble à l'image ci-contre, à droite.

Maintenant il nous faut paramétrer les gestionnaires d'événements pour tous les boutons que nous avons créés. Dans la zone de hiérarchie, sélectionnez le widget boBtnAjouter. Cela devrait surligner à la fois la ligne dans la hiérarchie et le bouton lui-même. Retournez à la section des attributs, sélectionnez l'onglet « Signaux » et déployez la ligne GtkToolButton pour afficher l'événement « clicked ». Choisissez comme précédemment « on_boBtnAjouter_clicked » comme gestionnaire de l'événement « clicked », puis cliquez au-dessus ou en dessous pour valider le changement. Faites cela pour tous les autres boutons que nous avons créés (en sélectionnant l'événement « on_boBtnSupprimer_clicked », etc.). Souvenez-vous de cliquer en dehors de la liste déroulante pour valider le changement, puis sauvegardez votre projet. Nos séparateurs n'ont pas besoin d'événements, ne les modifiez pas.

Ensuite, nous devons remplir nos « hbox ». Celle du haut contiendra (de gauche à droite) une étiquette, un widget texte et un bouton. Dans la boîte à outils, sélectionnez le widget « label » (pas le bleu) et placez-le dans l'emplacement de gauche ; puis placez un widget « saisie de texte » au milieu ; et enfin placez un bouton dans l'emplacement de droite. Faites la même chose pour la deuxième hbox.

Il faut maintenant régler les attributs pour les widgets que nous venons d'ajouter. Dans la zone de hiérarchie, sélectionnez label1 sous hbox1. Dans la section des attributs, sélectionnez l'onglet Général, descendez jusqu'à « Modifier l'apparence de l'étiquette » et saisissez « Chemin du fichier à sauvegarder : » comme texte d'étiquette. Puis allez sur l'onglet « Regroupement » et réglez « Déve-

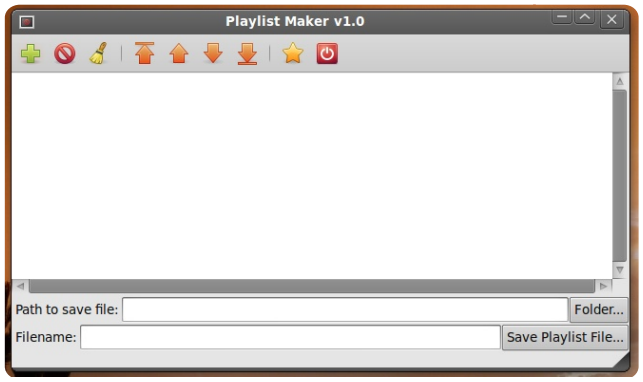




lopper » sur « Non ». Vous vous souvenez peut-être de la discussion du mois dernier au sujet du regroupement. Réglez le bourrage à 4, ce qui laissera un peu d'espace à gauche et à droite de l'étiquette. Maintenant, sélectionnez le bouton 1 et réglez également « Développer » sur « Non » sous l'onglet « Regroupement ». Retournez sur l'onglet Général et réglez le nom du bouton à « btnNomRepertoire ». Remarquez que le nom ne commence pas par « bo » car ce n'est pas un bouton de la barre d'outils. Descendez jusqu'à « Étiquette » et saisissez « Répertoire... ». Puis cliquez sur l'onglet « Signaux » et réglez l'événement du bouton à « on_btnNomRepertoire_clicked » dans GtkButton/clicked. Avant de régler les attributs du prochain ensemble de widgets dans l'autre hbox, il nous reste une chose à faire. Sélectionnez hbox1 dans la zone de hiérarchie et réglez « Développer » à « Non » sous l'onglet « Regroupement ». Cela permet à la hbox d'occuper moins d'espace. Pour terminer, réglez le nom du champ de saisie de texte à « txtChemin ».

```
<widget class="GtkWindow" id="FenetrePrincipale">
    <property name="visible">True</property>
    <property name="title" translatable="yes">Créateur de liste de lecture
v1.0</property>
    <property name="window_position">center</property>
    <property name="default_width">650</property>
    <property name="default_height">350</property>
    <signal name="destroy" handler="on_FenetrePrincipale_destroy"/>
```

Maintenant faites la même chose pour hbox2, en réglant « Développer » à « Non », puis le texte de l'étiquette à « Nom de fichier : », « Développer » à « Non » et « Bourrage » à 4. Réglez le nom du bouton à « btnSauvegarderListe », son texte à « Sauvegarder la liste de lecture... », son attribut « Développer » à « Non », réglez son événement « clicked » et réglez le nom du champ de saisie à « txtNomFichier ». Une fois de plus, sauvegardez tout.



Maintenant, notre fenêtre devrait ressembler à l'image ci-contre à gauche.

Tout cela est bien joli, mais qu'avons-nous fait exactement ? Nous ne pouvons pas exécuter cela comme un programme, puisque nous n'avons pas de code. Nous avons simplement créé un fichier XML nommé CreateurListeDeLecture.glade. Ne vous laissez pas tromper par l'extension : c'est réellement un fichier XML. Si vous faites attention, vous pouvez l'ouvrir dans votre éditeur de texte favori (gedit dans mon cas) et le consulter.

Vous verrez du texte qui décrit notre fenêtre et chacun des widgets avec ses attributs. Par exemple, regardons le code (en haut) pour le widget principal, la fenêtre elle-même.

Vous pouvez voir que le nom du widget est « FenetrePrincipale », son titre « Créateur de liste de lecture v1.0 », le gestionnaire d'événement, etc.

Regardons le code (en bas à droite) pour l'un de nos boutons de barre d'outils.

J'espère que cela commence à avoir du sens pour vous. Maintenant nous devons écrire du code pour nous permettre de voir notre beau travail faire vraiment quelque chose.

```
<child>
  <widget class="GtkToolButton" id="boBtnAjouter">
    <property name="visible">True</property>
    <property name="label" translatable="yes">Ajouter</property>
    <property name="use_underline">True</property>
    <property name="stock_id">gtk-add</property>
    <signal name="clicked" handler="on_boBtnAjouter_clicked"/>
  </widget>
  <packing>
    <property name="expand">False</property>
    <property name="homogeneous">True</property>
  </packing>
</child>
```

Ouvrez votre éditeur de code et commençons avec ceci (encadré jaune, miliieu de la première colonne).

Nous avons donc créé nos « import » à peu près comme le mois dernier. Remarquez que nous importons « sys » et « MP3 » depuis mutagen.mp3. Nous avons installé mutagen dans l'article numéro 9 ; si vous ne l'avez pas sur votre système reportez-vous à cet article. Nous aurons besoin de mutagen pour la prochaine fois, et de sys pour que le programme puisse se terminer proprement sur la dernière exception.

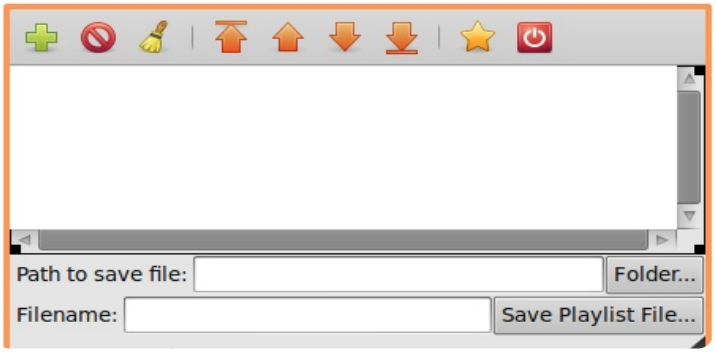
```
#!/usr/bin/env python
import sys
from mutagen.mp3 import MP3
try:
    import pygtk
    pygtk.require("2.0")
except:
    pass
try:
    import gtk
    import gtk.glade
except:
    sys.exit(1)
```

Ensuite, nous devons créer notre classe qui définira notre fenêtre : vous pouvez voir cela en haut à droite.

Nous avons déjà fait des choses très ressemblantes. Remarquez les deux dernières lignes : nous définissons le fichier glade (self.gladefile) en indiquant le nom du fichier que nous avons créé avec Glade Designer. Remarquez également que nous n'avons pas indiqué de chemin, juste un nom de fichier. Si votre fichier glade ne se situe pas au même endroit que votre code, vous devez indiquer ici un chemin ; cependant, il est toujours plus malin de les garder ensemble. Ensuite nous définissons notre fenêtre comme « self.wTree » : nous ferons appel à cela à chaque fois qu'on aura besoin de faire référence à la fenêtre. Nous précisons également que le fichier est au format XML, et que la fenêtre que nous utiliserons est celle qui s'appelle « FenetrePrincipale ». Vous pouvez avoir déclaré plusieurs fenêtres dans un seul fichier glade ; nous en reparlerons une autre fois.

```
class CreateurListeDeLecture:
    def __init__(self):
        #================================================
        #                    Creation de la fenetre
        #================================================
        self.gladefile = "CreateurListeDeLecture.glade"
        self.wTree =
gtk.glade.XML(self.gladefile,"FenetrePrincipale")
```

Maintenant il faut gérer les événements. Le mois dernier, nous avons utilisé les appels bouton.connect ou fenetre.connect pour faire référence à nos routines de gestion d'événements. Cette fois-ci, nous allons faire un peu différemment : nous allons utiliser un dictionnaire. Un dictionnaire est comme un tableau, sauf qu'au lieu d'utiliser un index entier, on utilise une clé pour accéder aux données. Clé et donnée : voici un morceau de code qui rendra sans doute cela plus compréhensible. Je ne vais vous montrer que deux événements pour l'instant (bas de la page à gauche).



Nous avons donc deux événements : « on_FenetrePrincipale_des-troy » et « on_boBtnQuitter_clicked » sont les clés de notre dictionnaire. Les données correspondantes sont « gtk.main_quit » pour les deux entrées. Lorsqu'un événement est levé par notre interface graphique, le système utilise cet événement pour trouver la clé dans notre dictionnaire, puis sait quelle routine appeler grâce aux données correspondantes. Maintenant nous devons connecter le dictionnaire au gestionnaire de signaux de notre fenêtre ; on fait cela avec la ligne de code suivante : self.wTree.signal_autoconnect(dict)

Nous sommes quasiment prêts. Il ne reste que la routine principale :

```
    #================================================
    #        Creation des gestionnaires d'evenements
    #================================================
    dict = {"on_FenetrePrincipale_destroy": gtk.main_quit,
            "on_boBtnQuitter_clicked": gtk.main_quit}
```

```
if __name__ == "__main__":
    cldl = CreateurListeDeLecture()
    gtk.main()
```

Sauvegardez ce fichier sous le nom « CreateurListeDeLecture.py ». Maintenant vous pouvez l'exécuter (voir page précédente, au milieu à droite).

Il ne fait pas grand chose pour l'instant, à part s'ouvrir et se fermer correctement. Nous verrons le reste la prochaine fois. Juste pour aiguiser votre appétit, nous discuterons de l'utilisation de la vue arborescente, des boîtes de dialogues et nous ajouterons pas mal de code.

Alors à la prochaine fois.

Fichier Glade :
http://fullcirclemagazine.pastebin.com/2NLaZ3yc

Source Python :
http://fullcirclemagazine.pastebin.com/dAqvxmba

EXTRA ! EXTRA ! LISEZ CECI

LE SERVEUR PARFAIT ÉDITION SPECIALE

Il s'agit d'une édition spéciale du Full Circle qui est une réédition directe des articles Le Serveur parfait qui ont déjà été publiés dans le FCM n° 31 à 34.

http://fullcirclemagazine.org/special-edition-1-the-perfect-server/

Des éditions spéciales du magazine Full Circle sont sorties dans un monde sans méfiance*

PYTHON ÉDITION SPECIALE n° 1

Il s'agit d'une reprise de Programmer en Python, parties 1 à 8 par Greg Walters.

http://fullcirclemagazine.org/python-special-edition-1/

* Ni Full Circle magazine, ni ses concepteurs ne s'excusent pour l'hystérie éventuellement causée par la sortie de ces publications.

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume quatre

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

Il ne s'agit de rien d'autre qu'une reprise de la série Programmer en Python, parties 22 à 26, numéros 48 à 52 ; pas de chichis, juste les faits.

Gardez à l'esprit la date de publication ; les versions actuelles du matériel et des logiciels peuvent être différentes de celles illustrées. Il vous est recommandé de bien vérifier la version de votre matériel et des logiciels avant d'essayer d'émuler les tutoriels dans ces numéros spéciaux. Il se peut que vous ayez des logiciels plus récents ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

SiteWeb :
http://www.fullcirclemagazine.org/

Forums :
http://ubuntuforums.org/forumdisplay.php?f=270

IRC : #fullcirclemagazine on chat.freenode.net

Équipe éditoriale :
Rédacteur en chef : Ronnie Tucker (pseudo : RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia (pseudo : admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcast : Robin Catling (pseudo : RobinCatling)
podcast@fullcirclemagazine.org
Dir. comm. : Robert Clipsham (pseudo : mrmonday) -
mrmonday@fullcirclemagazine.org

TUTORIEL

Programmer en Python - Partie 22

Greg Walters

CORRECTION

Dans la partie 21, il vous était dit de sauvegarder ce que vous aviez dans un fichier nommé « PlaylistMaker.glade » alors que, dans le code, il etait indiqué « playlistmaker.glade ». Je suis sûr que vous aviez remarqué que l'un contenait des majuscules et l'autre non. Le code s'exécutera seulement si il y a concordance entre le nom du fichier et l'appel, avec ou sans majuscule.

Pour bien commencer, vous devez avoir les fichiers playlistmaker.glade et playlistmaker.py du mois dernier. Si ce n'est pas le cas, sautez sur le numéro précédent pour les récupérer. Avant de passer au code, nous allons jeter un œil à ce qu'est un fichier de liste de lecture. Il y a plusieurs versions des listes de lecture, qui ont toutes des extensions différentes. Le fichier que nous allons créer sera de type *.m3u. Dans sa forme la plus simple, c'est juste un fichier texte qui commence par « #EXTM3U » et qui contient une entrée pour chaque fichier audio que vous voulez écouter - avec le chemin d'accès complet. Il y a aussi une extension qui peut être ajoutée avant chaque entrée contenant la longueur de la chanson, le nom de l'album d'où vient la chanson, le numéro de piste et le nom du morceau. Nous allons ignorer l'extension pour l'instant et nous concentrer uniquement sur la version de base. Voici un exemple d'un fichier de liste de lecture M3U :

```
#EXTM3U
Adult Contemporary/Chris Rea/Collection/02 - On The Beach.mp3
Adult Contemporary/Chris Rea/Collection/07 - Fool (If You Think It's Over).mp3
Adult Contemporary/Chris Rea/Collection/11 - Looking For The Summer.mp3
```

Tous les noms de chemins sont relatifs à l'emplacement du fichier de liste de lecture.

Bien... Maintenant passons au code. Vous voyez à droite le début du code source du mois dernier.

Maintenant, nous devons créer une routine de gestion d'événement pour chacun des événements que nous avons

```
##!/usr/bin/env python
import sys
from mutagen.mp3 import MP3
try:
  import pygtk
  pygtk.require("2.0")
except:
  pass
try:
  import gtk
  import gtk.glade
except:
  sys.exit(1)
```

puis la définition de la classe

```
class CreateurListeDeLecture:
  def __init__(self):
      self.gladefile = "CreateurListeDeLecture.glade"
      self.wTree = gtk.glade.XML(self.gladefile,"FenetrePrincipale")
```

et la routine principale

```
if __name__ == "__main__":
createurLDL = CreateurListeDeLecture()
gtk.main()
```

Ensuite, nous avons le dictionnaire qui devrait se trouver après la routine __init__.

```
def DicoEvenements(self):
  dict = {"on_FenetrePrincipale_destroy": gtk.main_quit,
    "on_boBtnQuitter_clicked": gtk.main_quit,
    "on_boBtnAjouter_clicked": self.on_boBtnAjouter_clicked,
    "on_boBtnSupprimer_clicked": self.on_boBtnSupprimer_clicked,
    "on_boBtnEffacer_clicked": self.on_boBtnEffacer_clicked,
    "on_boBtnHaut_clicked": self.on_boBtnHaut_clicked,
    "on_boBtnMonter_clicked": self.on_boBtnMonter_clicked,
    "on_boBtnDescendre_clicked": self.on_boBtnDescendre_clicked,
    "on_boBtnBas_clicked":self.on_boBtnBas_clicked,
    "on_boBtnAPropos_clicked": self.on_boBtnAPropos_clicked,
    "on_btnNomRepertoire_clicked": self.on_btnNomRepertoire_clicked,
    "on_btnSauvegarderListe_clicked":
self.on_btnSauvegarderListe_clicked}
  self.wTree.signal_autoconnect(dict)
```

mis en place. Notez que on_Fenetre-Principale_destroy et on_boBtnQuitter_clicked sont déjà faits pour nous, il n'en reste donc que dix autres à écrire (voir en haut à droite). Écrivons juste des ébauches pour l'instant.

Nous modifierons ces ébauches de routines dans quelques minutes. Pour l'instant, cela devrait nous permettre de démarrer l'application ; nous pourrons tester les choses au fur et à mesure que nous avançons. Nous devons quand même ajouter une ligne supplémentaire à la routine __init__ avant de pouvoir démarrer l'application. Après la ligne self.wTree, ajouter :

```
self.DicoEvenements()
```

Maintenant, vous pouvez exécuter l'application, voir la fenêtre, puis cliquer sur le bouton « Quitter de la barre d'outils » pour quitter l'application correctement. Enregistrez le code sous le nom « CreateurListeDeLecture-1a.py » et essayez-le. Souvenez-vous qu'il faut l'enregistrer dans le même dossier que le fichier glade que nous avons créé la dernière fois ou bien copier le fichier glade dans le dossier dans lequel vous avez enregistré ce code.

Nous avons également besoin de définir quelques variables pour une utilisation future. Ajoutez ceci après l'appel à DicoEvenements() dans la fonction __i- nit__.

```
self.CheminCourant = ""
self.LigneCourante = 0
self.NombreDeLignes = 0
```

Maintenant, nous allons créer une fonction qui nous permet d'afficher une boîte de dialogue à chaque fois que nous avons besoin de donner des informations à l'utilisateur. Il existe un ensemble de routines toutes faites que nous allons utiliser, mais nous allons faire une routine à nous pour nous faciliter les choses. C'est la routine gtk.MessageDialog et la syntaxe est la suivante :

```
gtk.MessageDialog (parent, drapeaux, MessageType, boutons, message)
```

Une discussion est nécessaire avant d'aller trop loin. Le type de message peut être l'un des suivants :

```
GTK_MESSAGE_INFO - message d'information
GTK_MESSAGE_WARNING - message d'avertissement
GTK_MESSAGE_QUESTION - question nécessitant un choix
GTK_MESSAGE_ERROR - message d'erreur fatale
```

Et les types de boutons sont :

```
def on_boBtnAjouter_clicked(self,widget):
        pass
def on_boBtnSupprimer_clicked(self,widget):
        pass
def on_boBtnEffacer_clicked(self,widget):
        pass
def on_boBtnHaut_clicked(self,widget):
        pass
def on_boBtnMonter_clicked(self,widget):
        pass
def on_boBtnDescendre_clicked(self,widget):
        pass
def on_boBtnBas_clicked(self,widget):
        pass
def on_boBtnAPropos_clicked(self,widget):
        pass
def on_btnNomRepertoire_clicked(self,widget):
        pass
def on_btnSauvegarderListe_clicked(self,widget):
        pass
```

```
GTK_BUTTONS_NONE - aucun bouton

GTK_BUTTONS_OK - un bouton OK
GTK_BUTTONS_CLOSE - un bouton Fermer
GTK_BUTTONS_CANCEL - un bouton Annuler
GTK_BUTTONS_YES_NO - boutons Oui et Non
GTK_BUTTONS_OK_CANCEL - boutons OK et Annuler
```

Normalement, vous utiliseriez le code suivant, ou du code similaire, pour créer la boîte de dialogue, l'afficher, attendre une réponse, puis la détruire.

```
dlg = gtk.MessageDialog (None, 0, gtk.MESSAGE_INFO, gtk.BUTTONS_OK, "Ceci est un message de test ...")
reponse = dlg.run ()
dlg.destroy ()
```

Toutefois, si vous voulez afficher une boîte de message plus d'une ou deux fois, c'est beaucoup de dactylographie. La règle générale est que si vous écrivez une série de lignes de code plus d'une ou deux fois, il est généralement préférable de créer une fonction puis de l'appeler. Pensez-y de cette manière : si nous voulons afficher un message de dialogue pour l'utilisateur, disons dix fois dans l'application, cela représente 10 x 3 (soit 30) lignes de code.

En faisant une fonction pour faire cela pour nous (en utilisant l'exemple que je viens de présenter), nous aurions 10 + 3 (soit 13) lignes de code à écrire. Plus nous appelons une boîte de dialogue, moins cela fait de code à taper, et plus lisible est notre code. Notre fonction (page suivante en haut à droite) nous permettra d'appeler l'un des quatre types de message de dialogue avec une seule routine en utilisant différents paramètres. C'est une fonction très simple que nous pourrons ensuite appeler comme suit :

```
self.MessageBox("info", "Le
bouton QUITTER a été cliqué")
```

Notez que si nous choisissons d'utiliser le type de dialogue MESSAGE_QUESTION, il y a deux réponses possibles qui seront retournées par la fenêtre de dialogue - un « oui » ou un « non ». Quel que soit le bouton cliqué par l'utilisateur, nous allons recevoir les informations de retour dans notre code. Pour utiliser la boîte de dialogue de question, l'appel ressemblera à ceci :

```
reponse = self.MessageBox(« ques-
tion », « Êtes-vous sûr de vou-
loir faire cela maintenant ? »)

if reponse == gtk.RESPONSE_YES:
```

```
def MessageBox(self,niveau,texte):
      if niveau == "info":
          dlg = gtk.MessageDialog(None,0,gtk.MESSAGE_INFO,gtk.BUTTONS_OK,texte)
      elif niveau == "warning":
          dlg = gtk.MessageDialog(None,0,gtk.MESSAGE_WARNING,gtk.BUTTONS_OK,texte)
      elif niveau == "error":
          dlg = gtk.MessageDialog(None,0,gtk.MESSAGE_ERROR,gtk.BUTTONS_OK,texte)
      elif niveau == "question":
          dlg = gtk.MessageDialog(None,0,gtk.MESSAGE_QUESTION,gtk.BUTTONS_YES_NO,texte)
      if niveau == "question":
          resp = dlg.run()
          dlg.destroy()
          return resp
      else:
          resp = dlg.run()
          dlg.destroy()
```

```
    print "clic sur oui"

elif reponse == gtk.RESPONSE_NO:

    print "clic sur non"
```

Vous voyez comment vous pouvez vérifier la valeur du bouton cliqué. Alors maintenant, remplacez l'appel à « pass » dans chacune de nos routines de gestion d'événement par ce que vous voyez ci-dessous à droite.

Nous n'allons pas le garder comme ça, mais cela vous donne une indication visuelle que les boutons fonctionnent comme nous le voulons. Enregistrez maintenant le code sous « Créateur-s'agit du widget treeview. Nous allons ListeDeLecture-1b.py » et testez votre programme. Maintenant nous allons

```
  def on_boBtnAjouter_clicked(self,widget):
       self.MessageBox("info","Clic sur bouton Ajouter...")
def on_boBtnSupprimer_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton Supprimer...")
def on_boBtnEffacer_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton Effacer...")
def on_boBtnHaut_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton Haut...")
def on_boBtnMonter_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton Monter...")
def on_boBtnDescendre_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton Descendre...")
def on_boBtnBas_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton Bas...")
def on_boBtnAPropos_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton À propos...")
def on_btnNomRepertoire_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton NomRepertoire...")
def on_btnSauvegarderListe_clicked(self,widget):
      self.MessageBox("info","Clic sur bouton SauvegarderListe...")
```

créer une fonction pour définir nos références de widgets. Cette routine va être appelée une seule fois, mais elle rendra notre code beaucoup plus maniable et lisible. En fait, nous voulons créer des variables locales

qui font référence à des widgets dans la fenêtre glade - afin que nous puissions faire appel à eux chaque fois que (et si jamais) nous en avons besoin. Mettez cette fonction (page suivante en haut à droite) en dessous de la fonction DicoEvenements.

Remarquez qu'il y a une chose qui n'est pas référencée dans notre routine. Il s'agit du widget treeview. Nous allons créer cette référence lorsque nous créerons l'arborescence elle-même. Notez également la dernière ligne de notre routine. Pour utiliser la barre d'état, il faut s'y référer par son id de contexte. Nous allons utiliser cela plus loin. Ensuite, nous allons mettre en place la fonction qui affiche le dialogue « à propos » quand on clique sur le bouton À propos de la barre d'outils. Encore une fois, ceci est une routine intégrée fournie par la bibliothèque GTK. Placez ceci après la fonction MessageBox. Voici le code, en bas à droite. Sauvegardez votre code, puis faites un essai. Vous devriez voir une fenêtre pop-up, centrée dans notre application, qui affiche ce que nous avons prévu. Il y a plusieurs attributs que vous pouvez définir pour la boîte à propos (qui peuvent être trouvés sur http://www.pygtk.org/docs/pygtk/class-gtkaboutdialog.html), mais ceux-ci sont ceux que je considère être un ensemble minimal.

```
def ReferencesWidgets(self):
     self.txtNomFicher = self.wTree.get_widget("txtNomFicher")
    self.txtChemin = self.wTree.get_widget("txtChemin")
    self.boBtnAjouter = self.wTree.get_widget("boBtnAjouter")
    self.boBtnSupprimer = self.wTree.get_widget("boBtnSupprimer")
    self.boBtnEffacer = self.wTree.get_widget("boBtnEffacer")
    self.boBtnQuitter = self.wTree.get_widget("boBtnQuitter")
    self.boBtnAPropos = self.wTree.get_widget("boBtnAPropos")
    self.boBtnHaut = self.wTree.get_widget("boBtnHaut")
    self.boBtnMonter = self.wTree.get_widget("boBtnMonter")
    self.boBtnDescendre = self.wTree.get_widget("boBtnDescendre")
    self.boBtnBas = self.wTree.get_widget("boBtnBas")
    self.btnNomRepertoire = self.wTree.get_widget("btnNomRepertoire")
    self.btnSauvegarderListe = self.wTree.get_widget("btnSauvegarderListe")
    self.sbar = self.wTree.get_widget("statusbar3")
    self.context_id = self.sbar.get_context_id("Statusbar")
```

puis ajoutez un appel à ceci juste après l'appel à self.DicoEvenements() dans la routine __init__.

```
    self.ReferencesWidgets()
```

Avant de poursuivre, nous devons discuter de ce qui se produira à partir d'ici. L'idée générale est que l'utilisateur clique sur le bouton « Ajouter » de la barre d'outils, nous afficherons alors une boîte de dialogue de fichier pour lui permettre d'ajouter des fichiers à la liste de lecture, puis nous afficherons les informations du fichier dans notre widget treeview. De là, il peut ajouter d'autres fichiers, supprimer un fichier unique, supprimer tous les fichiers, déplacer un fichier vers le haut ou le

```
def AfficherAPropos(self):
      apropos = gtk.AboutDialog()
      apropos.set_program_name("Createur de liste de
lecture")
      apropos.set_version("1.0")
      apropos.set_copyright("(c) 2011 by Greg Walters")
      apropos.set_comments("Ecrit pour le Full Circle
Magazine")
      apropos.set_website("http://thedesignatedgeek.com")
      apropos.run()
      apropos.destroy()
```

Maintenant, commentez (ou retirez simplement) l'appel à MessageBox dans la routine on_boBtnAPropos_clicked, et remplacez-le par un appel à la fonction AfficherAPropos. Cela devrait ressembler à :

```
def on_boBtnAPropos_clicked(self,widget):
    #self.MessageBox("info","Clic sur bouton APropos...")
    self.AfficherAPropos()
```

bas, ou bien tout en haut ou tout en bas de l'arborescence. Enfin, il va définir le chemin où le fichier sera enregistré, fournir un nom de fichier avec une extension « m3u », puis cliquer sur le bouton « Sauvegarder ». Bien que cela semble assez simple, il se passe beaucoup de choses en coulisses. La magie se produit dans le widget treeview, nous allons donc en discuter. Cela ira assez loin, alors lisez attentivement, car il faut le comprendre pour éviter de commettre des erreurs plus tard. Une arborescence peut être quelque chose d'aussi simple qu'une liste à colonnes de données, comme dans une feuille de calcul ou une base de données, ou bien elle peut être plus complexe, comme une liste de fichiers/dossiers avec des parents et enfants, où le dossier serait le parent et les fichiers de ce dossier seraient les enfants, ou quelque chose d'encore plus complexe. Pour ce projet, nous allons utiliser le premier exemple, une liste à colonnes. Dans la liste, il y aura trois colonnes. Une pour le nom du fichier de musique, une pour l'extension du fichier (mp3, ogg, wav, etc.) et la dernière colonne pour le chemin d'accès. En combinant tout ça dans une chaîne (chemin d'accès, nom de fichier, extension) on obtient l'entrée que nous allons écrire dans la liste de lecture. Vous pourriez bien sûr ajouter d'autres colonnes si vous le souhaitez, mais pour l'instant nous allons nous contenter de trois. Une arborescence est simplement un conteneur visuel de stockage qui détient et affiche un modèle. Le modèle est le véritable « dispositif » qui contient et manipule nos données. Il existe deux modèles prédéfinis qui sont utilisés avec un treeview, mais vous pouvez certainement créer le vôtre. Cela étant dit, pour 98 % de votre travail, l'un des deux modèles prédéfinis fera ce dont vous avez besoin. Les deux types sont GTKListStore et GTKTreeStore. Comme leur nom l'indique, le modèle ListStore est habituellement utilisé pour les listes, le TreeStore est utilisé pour les arbres. Pour notre application, nous allons utiliser un GTKListStore. Les étapes de base sont les suivantes :

```
def SetupTreeview(self):
    self.cNomFic = 0
    self.cTypeFic = 1
    self.cCheminFic = 2
    self.sNomFic = "NomFichier"
    self.sTypeFic = "Type"
    self.sCheminFic = "Dossier"
    self.treeview = self.wTree.get_widget("treeview1")
    self.AjouterColonne(self.sNomFic,self.cNomFic)
    self.AjouterColonne(self.sTypeFic,self.cTypeFic)
    self.AjouterColonne(self.sCheminFic,self.cCheminFic)
    self.listeLecture = gtk.ListStore(str,str,str)
    self.treeview.set_model(self.listeLecture)
    self.treeview.set_grid_lines(gtk.TREE_VIEW_GRID_LINES_BOTH)
```

- Créer une référence au widget TreeView.
- Ajouter les colonnes.
- Définir le type de moteur de rendu à utiliser.
- Créer le ListStore.
- Définir l'attribut de modèle dans l'arborescence de notre modèle.
- Remplir les données.

La troisième étape consiste à mettre en place le type de moteur de rendu que la colonne utilisera pour afficher les données. C'est tout simplement une routine qui est utilisée pour tracer les données dans le modèle de l'arbre. GTK fournit de nombreux moteurs de rendu de cellules différents, mais normalement vous utiliserez le plus souvent GtkCellRenderText et GtkCellRendererToggle.

Nous allons donc créer une fonction (ci-dessus) qui met en place notre widget TreeView. Nous allons l'appeler SetupTreeview. Nous allons d'abord définir quelques variables pour nos colonnes, définir la variable de référence du TreeView proprement dit, ajouter les colonnes, mettre en place le ListStore, et définir le modèle. Voici le code pour la fonction. Placez-le après la fonction ReferencesWidget.

Les variables cNomFic, cTypeFic et cCheminFic définissent les numéros de colonne. Les variables sNomFic, sTypeFic et sCheminFic contiennent les noms de colonnes de notre vue. La septième ligne définit la variable de référence du widget treeview tel qu'il figure dans notre fichier glade.

Ensuite nous appelons une routine (page suivante, en haut à droite), que nous allons créer dans un instant, pour chaque colonne que nous voulons. Puis, nous définissons notre GTKListStore avec trois champs de texte et, enfin, nous utilisons ce GTKListStore comme attribut de

modèle de notre widget TreeView. Nous allons ensuite créer la fonction AjouterColonne. Placez-la après la fonction SetupTreeview.

Chaque colonne est créée avec cette fonction. Nous lui passons le titre de la colonne (ce qui est affiché sur la première ligne de chaque colonne) et un idColonne. Dans ce cas, nous utilisons les variables que nous avons créées plus tôt (sNomFic et cNomFic). Nous créons ensuite une colonne dans notre widget TreeView donnant le titre, le type de rendu de cellule et enfin l'id de la colonne. Nous indiquons ensuite que la colonne est redimensionnable, nous définissons l'id de tri et ajoutons enfin la colonne dans le TreeView.

Ajoutez ces deux fonctions à votre code. J'ai choisi de les mettre tout de suite après la fonction ReferencesWidget, mais vous pouvez les mettre n'importe où dans la classe CreateurListeDeLecture. Ajoutez la ligne suivante après l'appel à ReferencesWidget() dans la fonction __init__ pour appeler la fonction :

```
self.SetupTreeview ()
```

Enregistrez et exécutez votre programme et vous verrez que nous avons maintenant trois colonnes avec en-têtes dans notre widget TreeView.

```
def AjouterColonne(self,titre,idColonne):
    colonne = gtk.TreeViewColumn(titre,gtk.CellRendererText(),text=idColonne)
    colonne.set_resizable(True)
    colonne.set_sort_column_id(idColonne)
    self.treeview.append_column(colonne)
```

Il reste tellement de choses à faire. Nous devons avoir un moyen d'obtenir les noms de fichiers de musique de l'utilisateur et un moyen de les mettre dans le TreeView sous forme de lignes de données. Nous devons créer nos fonctions Supprimer, Effacer tout, les fonctions de déplacement, la routine de sauvegarde et les routines de chemins de fichiers, plus quelques « jolies » choses qui donneront à notre application un aspect plus professionnel. Commençons par la routine « Ajouter ». Après tout, c'est le premier bouton sur notre barre d'outils. Lorsque l'utilisateur clique sur le bouton Ajouter, nous voulons faire apparaître une fenêtre de dialogue « standard » d'ouverture de fichier, qui permet des sélections multiples. Une fois que l'utilisateur a fait son choix, nous voulons ensuite prendre ces données et les ajouter dans l'arborescence, comme je l'ai indiqué ci-dessus. Ainsi, la première chose logique à faire est de travailler sur la boîte de dialogue Fichier. Encore une fois, GTK nous fournit un moyen d'appeler une boîte de dialogue « standard » de fichiers. Nous pourrions coder ça en dur simplement avec des lignes de code dans le gestionnaire d'événements on_boBtnAjouter_clicked, mais nous allons faire une classe distincte pour le gérer. Tant que nous y sommes, nous pouvons faire en sorte que cette classe gère non seulement un dialogue Ouvrir un fichier, mais aussi un dialogue Sélectionner un dossier. Comme auparavant avec la fonction MessageBox, vous pouvez l'extraire dans un fichier qui contient toutes sortes de routines réutilisables pour un usage ultérieur.

Nous allons commencer par définir une nouvelle classe appelée DialogueFichier qui a une seule fonction appelée AfficheDialogue. Cette fonction prendra deux paramètres, l'un appelé « type » (un '0' ou un '1'), qui précise si nous créons un dialogue d'ouverture de fichier ou de sélection de dossier, et l'autre, qui est le chemin à utiliser pour la vue par défaut de la boîte de dialogue appelée CheminCourant. Créez cette classe juste avant notre code principal à la fin du fichier source.

```
class DialogueFichier:
  def AfficheDialogue(self,
type,CheminCourant):
```

La première partie de notre code doit être une instruction IF

```
if type == 0: # choix de fichier
  ...
else: # choix de dossier
  ...
```

Avant d'aller plus loin, nous allons voir la façon dont la boîte de dialogue de fichier/dossier est effectivement appelée et utilisée. La syntaxe de la boîte de dialogue se présente comme suit :

```
gtk.FileChooserDialog(titre,parent,action,boutons,backend)
```

et retourne un objet fenêtre de dialogue. Notre première ligne (dans le cas où type vaut 0) sera la ligne ci-dessous.

Comme vous pouvez le voir, le titre est « Choisir les fichiers a ajouter… », le parent est défini sur none (aucun). Nous demandons une fenêtre de type ouverture de fichier (action) et nous

voulons des boutons « Annuler » et « Ouvrir », les deux utilisant des icônes de type « stock ». Nous réglons également les codes de retour de gtk.RESPONSE_CANCEL et gtk.RESPONSE_OK lorsque l'utilisateur fait ses choix. L'appel au sélecteur de dossier dans la clause else est similaire.

Fondamentalement, les seules choses qui ont changé entre les deux définitions sont le titre (ci-dessus à droite) et le type d'action. Donc le code de la classe devrait maintenant être le code affiché au milieu à droite.

Nous définissons la réponse par défaut à la touche OK, puis activons la fonctionnalité de sélection multiple pour que l'utilisateur puisse sélectionner (vous l'aurez deviné) plusieurs fichiers à ajouter. Si nous n'avions pas indiqué cela, la boîte de dialogue permettrait seulement de sélectionner un fichier à la fois, car set_select_multiple est réglé sur faux par défaut. Nos lignes sui-vantes règlent le chemin actuel, puis affichent la boîte de dialogue elle-même. Avant de taper le code, je vais vous expliquer pourquoi nous devons nous occuper du chemin courant. À chaque fois que vous faites apparaître une boîte de dialogue de fichier et que vous ne définissez pas un chemin, la valeur par défaut est le dossier où réside notre application. Ainsi, si les fichiers de musique que l'utilisateur utilise sont dans /media/musique/ ils sont ensuite triés par genre puis par artiste, et puis après par album. Supposons également que l'utilisateur a installé notre application dans /home/user2/createurListeDeLecture. Chaque fois que nous faisons apparaître le dialogue, le dossier de départ serait /home/user2/createurListeDeLecture. Rapidement, l'utilisateur devrait se sentir frustré par cela, préférant retrouver le dernier dossier dans lequel il était lorsqu'il démarre la prochaine fois. Vous comprenez ? Bien. Voici donc en bas à droite les lignes de code suivantes.Ici, nous vérifions les réponses renvoyées. Si l'utilisateur a cliqué sur le bouton « Ouvrir » qui renvoie gtk.RES-PONSE_OK, nous obtenons le nom ou les noms des fichiers que l'utilisateur a sélectionné, on définit le chemin d'accès courant vers le dossier où nous sommes, on détruit la boîte de dialogue, puis on renvoie les données à la routine appelante. Si, en revanche, l'utilisateur a cliqué sur le bouton « Annuler », il suffit de détruire la boîte de dialogue. Je mets l'instruction print là juste pour vous montrer que l'appui sur le bouton a fonctionné. Vous pouvez la laisser ou la retirer. Notez que lorsque nous sortons de la partie concernant le bouton Ouvrir dans cette routine, nous renvoyons deux ensembles de valeurs : selectionFichiers qui est une liste des fichiers sélectionnés par l'utilisateur, ainsi que le CheminCourant.

Afin que la routine fasse quelque chose, ajoutez la ligne suivante dans la routine on_boBtnAjouter_clicked :

```
fd = DialogueFichier ()

fichiersChoisis,self.CheminCou-
rant = fd.AfficheDia-
logue(0,self.CheminCourant)
```

Ici on récupère les deux valeurs de retour qui sont renvoyées depuis le return. Pour le moment, ajoutez le code ci-dessous pour voir à quoi les informations retournées ressemblent :

```
for f in fichiersChoisis:
  print "Choix utilisateur :
%s" % f

print “Chemin courant : %s” %
self.CheminCourant
```

Lorsque vous exécutez le programme, cliquez sur le bouton « Ajouter ». Vous verrez la boîte de dialogue de fichier. Allez maintenant à un endroit où vous avez des fichiers et sélectionnez-les. Vous pouvez appuyer sur la touche [Ctrl] et cliquer sur plusieurs fichiers pour les sélectionner individuellement, ou sur la touche [Maj] pour sélectionner plusieurs fichiers contigus. Cliquez sur le bouton « Ouvrir », et examinez la réponse dans un terminal. Remarquez que si vous cliquez sur le bouton « Annuler » à ce moment, vous obtiendrez un message d'erreur. C'est parce que le code ci-dessus suppose qu'il n'y a pas de fichiers sélectionnés. Ne vous inquiétez pas pour l'instant, nous allons régler cela sous peu. Je voulais simplement vous permettre de voir ce qui revient si l'on appuie sur le bouton « Ouvrir ». Une chose que nous devrions faire est d'ajouter un filtre à notre fenêtre d'ouverture de fichier. Puisque nous attendons que l'utilisateur sélectionne normalement des fichiers de musique, nous devrions : 1) donner la possibilité d'afficher des fichiers de musique uniquement et, 2) donner la possibilité d'afficher tous les fichiers au cas où. Nous faisons cela en utilisant les attributs FileFilter de la boîte de dialogue. Voici le code pour cela, qu'il faut placer dans la partie « type == 0 » juste après la ligne créant le dialogue.

```
filtre = gtk.FileFilter()
filtre.set_name(“Fichiers musi-
caux”)
filtre.add_pattern(“*.mp3”)
```

```
filtre.add_pattern("*.ogg")
filtre.add_pattern("*.wav")d
ialogue.add_filter(filtre)
filtre =
gtk.FileFilter()filtre.set_n
ame("Tous les fichiers")
filtre.add_pattern("*")dialo
gue.add_filter(filtre)
```

Nous mettons en place deux « groupes », l'un pour les fichiers de musique (filtre.set_name("Fichiers musicaux")), et l'autre pour tous les fichiers. Nous utilisons un motif pour définir les types de fichiers que nous voulons. J'ai défini trois motifs, mais vous pouvez ajouter ou supprimer tous ceux que vous souhaitez. Je mets le filtre pour la musique en premier, puisque c'est ce qui intéresse principalement l'utilisateur. Ainsi, les étapes sont :

- Définir une variable de filtre.
- Régler le nom.
- Ajouter un motif.
- Ajouter le filtre à la boîte de dialogue.

Vous pouvez avoir autant ou aussi peu de filtres que vous le souhaitez. Notez également qu'une fois que vous avez ajouté le filtre à la boîte de dialogue, vous pouvez réutiliser la variable de filtre. Retournez dans la routine on_boBtnAjouter_clicked, commentez les dernières lignes que nous avons ajoutées et remplacez-les par cette seule ligne :

```
self.AjouterFichiers(fichiers-
Choisis)
```

Notre routine ressemble maintenant au code affiché ci-contre.

Ainsi, lorsque nous aurons la réponse au retour de la fenêtre de sélection de fichiers, nous enverrons la liste contenant les fichiers sélectionnés à cette routine. Une fois ici, nous créons une variable de compteur (le nombre de fichiers que nous ajoutons), puis analysons la liste. Rappelez-vous que chaque entrée contient le nom de fichier complet avec le chemin et l'extension. Nous allons devoir fractionner le nom du fichier en chemin, nom de fichier et extension. Nous récupérons d'abord le tout dernier « . » dans le nom de fichier et supposons que c'est le début de l'extension, et nous affectons sa position dans la chaîne à debutExt. Nous trouvons ensuite le tout dernier « / » dans le nom du fichier pour déterminer le début du nom de fichier. Puis, nous découpons la chaîne en extension, nom de fichier et chemin du fichier. Nous plaçons ensuite ces valeurs dans une liste nommée « data » et ajoutons ceci dans listeLecture. Nous incrémentons le compteur puisque nous avons fait tout le travail. Enfin on incrémente la variable NombreDeLignes qui contient le nombre total de lignes dans listeLecture et nous affichons un message dans la barre d'état.

```
  def on_boBtnAjouter_clicked(self,widget):
      fd = DialogueFichier()
      fichiersChoisis,self.CheminCourant = fd.AfficheDia-
logue(0,self.CheminCourant)
      self.AjouterFichiers(fichiersChoisis)
```

Nous devons maintenant créer la fonction à laquelle nous venons de faire appel. Placez cette fonction après la routine on_btnSauvegarderListe_clicked.

```
  def AjouterFichiers(self,ListeFichiers):
      compteur = 0
      for f in ListeFichiers:
          debutExt = f.rfind(".")
          debutnomFic = f.rfind("/")
          extension = f[debutExt+1:]
          nomFic = f[debutnomFic+1:debutExt]
          cheminFic = f[:debutnomFic]
          data = [nomFic,extension,cheminFic]
          self.listeLecture.append(data)
          compteur += 1
      self.NombreDeLignes += compteur
      self.sbar.push(self.context_id,"%s fichiers ajoutes
sur un total de %d" % (compteur,self.NombreDeLignes))
```

Maintenant vous pouvez lancer l'application et voir les données dans l'arborescence. Comme toujours, le code complet peut être trouvé ici :

http://pastebin.com/wTCcGDSW.

La prochaine fois, nous allons finaliser notre application, en remplissant les routines manquantes, etc.

TUTORIEL

Programmer en Python - Partie 23

Greg Walters

Cette fois-ci, nous allons terminer notre programme de création de liste de lecture. La dernière fois, nous avions bien avancé, mais nous n'avons pas terminé certaines parties. Nous ne pouvons pas encore sauvegarder la liste de lecture, les fonctions de déplacement ne sont pas implémentées, nous ne pouvons pas choisir le chemin vers lequel on veut sauvegarder, etc. Cependant, nous devons faire certaines choses avant de commencer à coder. Tout d'abord, nous devons trouver une image pour le logo de notre application dans la boîte « À propos » et lorsque l'application est minimisée. Vous pouvez chercher une icône qui vous plaît dans le répertoire /usr/share/icons, ou aller sur le web en chercher une ou encore en créer une vous-même. Quel que soit le choix, placez cette image dans le répertoire contenant le code source et le fichier glade du mois dernier. Nommez-la logo.png. Ensuite, nous devons ouvrir le fichier glade du mois dernier et faire quelques changements.

Tout d'abord, avec la FenetrePrincipale, allez dans l'onglet Général et descendez jusqu'à trouver Icône. En utilisant l'outil de parcours de fichiers, trouvez votre icône et sélectionnez-la. Maintenant le champ de texte devrait contenir « logo.png ». Puis, dans la boîte de hiérarchie, choisissez treeview1, allez dans l'onglet Signaux et ajoutez un gestionnaire pour on_treeview1_cur-sor_changed dans la partie GtkTreeView | cursor-changed. Souvenez-vous que nous avons vu le mois dernier que vous devez cliquer à côté pour conserver vos modifications. Enfin, toujours dans la boîte hiérarchie, choisissez txtNomFichier et allez dans l'onglet Signaux. Descendez jusqu'à trouver GtkWidget et descendez encore jusqu'à key_press_event. Ajoutez un gestionnaire d'événement pour on_txtNomFichier_key_press_event. Sauvegardez votre projet glade et fermez glade.

Maintenant il est temps de terminer notre projet. Nous commencerons à coder là où nous en étions restés le mois dernier.

La première chose que je veux faire est modifier le code de la classe DialogueFichier. Si vous vous souvenez de la dernière fois, si l'utilisateur cliquait le bouton « Annuler », il se produisait une erreur. Nous allons commencer par corriger ça. À la fin de la routine, vous avez le code ci-dessus.

```
elif response == gtk.RESPONSE_CANCEL:
     print 'Annulation, aucun fichier choisi'
     dialog.destroy()
```

Remarquez que nous ne renvoyons rien. C'est ce qui causait l'erreur. Pour réparer cela, ajoutez la ligne suivante après la ligne dialog.destroy() :

```
Return ([],"")
```

Ainsi il n'y aura plus d'erreur. Ensuite, ajoutons le gestionnaire d'événement que nous avons créé dans glade pour le champ de texte. Dans notre dictionnaire, ajoutez la ligne suivante :

```
"on_txtNomFichierFilename_key_press_event":
self.txtNomFichierKeyPress,
```

Vous vous souvenez que cela crée une fonction pour gérer l'appui sur les touches du clavier. Créons maintenant la fonction :

```
def txtNomFichierKeyPress(self,widget,data):
   if data.keyval == 65293: # valeur de la touche Entree
      self.SauveListeLecture()
```

Comme vous pouvez le supposer, cela regarde simplement la valeur de chaque touche enfoncée lorsque l'utilisateur se trouve dans le champ de texte txtNomFichier et la compare à la valeur 65293, qui est le code attribué à la touche Entrée. Si cela correspond, alors il appelle la fonction SauvegarderListe. L'utilisateur n'a même pas besoin de cliquer sur le bouton.

Maintenant passons au code. Occu-

pons-nous du bouton « Effacer » de la barre d'outils. Lorsque l'utilisateur clique sur ce bouton, on veut effacer la liste arborescente et ListStore. Cela se fait en une ligne, que l'on peut placer dans la routine on_boBtnEffacer_clicked.

```
def
on_boBtnEffacer_clicked(self
,widget) ::

    self.playList.clear()
```

Nous disons simplement à la liste de lecture ListStore de s'effacer. C'était facile. Maintenant occupons-nous du bouton « Supprimer » de la barre d'outils. C'est plus difficile, mais une fois terminé vous allez comprendre.

D'abord nous devons parler de la façon dont nous récupérons une sélection depuis la liste arborescente et ListStore. C'est un peu compliqué, alors allons doucement. Pour récupérer des données depuis ListStore, nous devons d'abord récupérer un objet gtk.TreeSelection qui nous aidera à gérer la sélection à l'intérieur d'un treeview. Ensuite, on utilise cet objet pour récupérer le type de modèle et un itérateur qui contient les lignes sélectionnées.

Je sais que vous pensez : « Mais bon sang, qu'est-ce qu'un itérateur ? ». Eh bien, vous en avez déjà utilisé sans même le savoir. Regardez le code suivant (ci-dessus à droite) provenant de la fonction AjouterFichiers du mois dernier.

Regardez la boucle for. On utilise un itérateur pour parcourir la liste ListeFichiers. Dans ce cas, l'itérateur passe tout simplement d'une entrée de la liste à la suivante, renvoyant chaque élément séparément. Nous allons créer un itérateur, le remplir avec les lignes de la vue arborescente sélectionnées et l'utiliser comme une liste. Voici donc le code (au milieu à droite) pour on_boBtnSupprimer.

La première ligne crée l'objet TreeSelection. On l'utilise pour récupérer les lignes sélectionnées (il n'y en a qu'une car notre modèle n'est pas réglé pour offrir la sélection multiple), remplir une liste nommée iter avec, et la parcourir en enlevant chaque élément (comme la méthode .clear). On décrémente également la variable NombreDeLignes, puis on affiche le nombre de fichiers dans la barre d'état.

Maintenant, avant de passer aux fonctions de déplacement, occupons-nous de la fonction de sauvegarde du chemin des fichiers. On utilisera notre classe DialogueFichier comme précédemment. On placera tout le code pour faire cela (en bas à droite) dans la routine on_boBtnNomRepertoire_clicked.

La seule chose vraiment différente par rapport à avant est la dernière ligne de ce code. On place le nom du chemin retourné par la fenêtre de dialogue dans le champ de texte que l'on a précédemment initialisé avec la méthode set_text. Souvenez-vous que les données nous sont renvoyées sous forme de liste, même s'il n'y a qu'un seul élément. C'est pourquoi

```
def AjouterFichiers(self,ListeFichiers):
    compteur = 0
    for f in ListeFichiers:
        debutExt = f.rfind(".")
        debutnomFic = f.rfind("/")
        extension = f[debutExt+1:]
        nomFic = f[debutnomFic+1:debutExt]
        cheminFic = f[:debutnomFic]
        data = [nomFic,extension,cheminFic]
        self.listeLecture.append(data)
        compteur += 1
```

```
def on_boBtnSupprimer_clicked(self,widget):
    sel = self.treeview.get_selection()
    (modele,lignes) = sel.get_selected_rows()    iter=[]
    for ligne in lignes:
      iter.append(self.listeLecture.get_iter(ligne))
    for i in iter:
      if i is not None:
        self.listeLecture.remove(i)
        self.NombreDeLignes -= 1
    self.sbar.push(self.context_id,"%d fichiers dans la
liste." % (self.NombreDeLignes))
```

```
def on_btnNomRepertoire_clicked(self,widget):
    fd = DialogueFichier()
    cheminFichier,self.CheminCourant =
fd.AfficheDialogue(1,self.CheminCourant)
    self.txtChemin.set_text(cheminFichier[0])
```

on utilise chemin[0].

Écrivons la fonction de sauve-garde de fichier. On peut faire ça avant de passer aux fonctions de déplacement. Nous allons créer une fonc-tion SauvegarderListe. La première chose à faire (ci-dessus à droite) est de vérifier s'il y a quelque chose dans le champ de texte txtChemin. Ensuite nous devons vérifier s'il y a un nom de fichier dans le champ de texte txtNomFi-chier. Pour ces deux valeurs, on utilise la méthode get_text() du champ de texte.

Maintenant que l'on a un chemin (cf) et un nom de fichier (nf), on peut ouvrir le fichier, imprimer notre entête M3U et parcourir la liste de lecture. Le chemin est stocké (si vous vous souvenez) dans la colonne 2, le nom du fichier dans la colonne 0 et l'extension dans la colonne 1. On crée simplement (à droite) une chaîne, puis on l'écrit dans le fichier et enfin on ferme le fichier.

On peut maintenant commencer à travailler sur les fonctions de déplacement. Commençons par la routine Haut. Comme nous l'avons fait en écrivant la fonction Supprimer, on

```
def SauveListeLecture(self):
      cf = self.txtChemin.get_text()      # recuperer le chemin dans le champ de texte
      nf = self.txtNomFichier.get_text() # recuperer le nom du fichier dans le champ de
texte
```

Maintenant on vérifie les valeurs :

```
          if cf == "":                        # SI le chemin est vide
              self.MessageBox("erreur","Veuillez fournir un chemin pour la liste de lecture.")
      elif nf == "":                      # SI le nom de fichier est vide
          self.MessageBox("erreur","Veuillez fournir un nom pour le fichier liste de
lecture.")
      else:                               # Sinon, on peut continuer
```

```
fic = open(cf + "/" + nf,"w")  # ouvrir le fichier
fic.writelines('#EXTM3U\n')    # afficher l'en-tete M3U
for ligne in self.listeLecture:
    fic.writelines("%s/%s.%s\n" % (ligne[2],ligne[0],ligne[1])) # ecrit les donnees
fic.close   # referme le fichier
```

Enfin, on affiche un message informant l'utilisateur que le fichier est sauvegardé.

```
self.MessageBox("info","La liste de lecture est sauvegardee !")
```

On doit maintenant appeler cette routine depuis notre routine de gestion d'événement on_btnSauvegarderListe_clicked.

```
def on_btnSauvegarderListe_clicked(self,widget):
     self.SauveListeLecture()
```

Sauvegardez votre code et testez-le. Votre liste de lecture devrait être sauvegardée correctement et ressembler à l'exemple que je vous ai montré le mois dernier.

récupère la sélection puis la ligne sélectionnée. Ensuite on doit parcourir les lignes pour récupérer 2 variables. Nous les appellerons chemin1 et chemin2. chemin2 sera réglé à 0 dans ce cas, car c'est la ligne de «_destination ». chemin1 est la ligne

```
ef on_boBtnHaut_clicked(self,widget):
        sel = self.treeview.get_selection()
        (modele,lignes) = sel.get_selected_rows()
        for chemin1 in lignes:
            chemin2 = 0
        iter1=modele.get_iter(chemin1)
        iter2 = modele.get_iter(chemin2)
        modele.move_before(iter1,iter2)
```

que l'utilisateur a sélectionnée. On utilise enfin la méthode modele.move_before() pour déplacer la ligne sélectionnée sur la ligne 0, en poussant d'office tout vers le bas. Nous placerons le code (ci-contre à droite) directement dans la routine on_boBtn Haut_clicked.

Pour la fonction Bas, nous utiliserons presque le même code que pour la routine Haut, mais au lieu d'utiliser la méthode modele.moveBefore(), nous utiliserons la méthode modele.moveAf-ter() et, au lieu de régler chemin2 à 0, on le réglera à self.NombreDeLignes-1. Maintenant vous comprenez à quoi sert la variable NombreDeLignes. Souvenez-vous que les lignes sont numérotées à partir de 0, donc il faut utiliser NombreDeLignes-1 (en haut à droite).

Maintenant regardons ce que donne la fonction Monter. À nouveau, elle est très ressemblante aux deux fonctions que nous venons de créer. Cette fois-ci, on a chemin1 qui contient la ligne sélectionnée, et on règle chemin2 à NumeroLigne-1. Ensuite, SI chemin2 (la ligne de destination) est supérieur ou égal à 0, on utilise la méthode modele.swap() (au milieu à droite).

C'est la même chose pour la fonction Descendre. Cette fois-ci, on vérifie que chemin2 est plus PETIT ou égal à self.NombreDeLignes-1 (en bas à droite).

Maintenant, modifions quelques fonctionnalités de notre liste de lecture. Dans l'article du mois dernier, je vous ai montré le format de base d'une liste de lecture (en bas).

Cependant, je vous ai indiqué qu'il y avait aussi un format étendu. Dans le format étendu, il y a une ligne supplémentaire que l'on peut ajouter au fichier avant chaque chanson, contenant des informations supplémentaires sur la chanson. Le format de cette ligne est le suivant :

```
#EXTINF:[longueur de la chanson en secondes],[Nom de l'artiste] – [Titre de la chanson]
```

Vous vous demandiez peut-être pourquoi on a inclus la bibliothèque mutagen depuis le début alors qu'on ne l'a jamais utilisée. Eh bien, nous allons l'utiliser maintenant. Pour vous rafraîchir la mémoire, la bibliothèque mutagen permet d'avoir accès aux informations des balises ID3 des fichiers MP3. Pour lire la discussion complète là-dessus, reportez-vous au numéro 35 du Full Circle qui contient la partie 9 de cette série. Nous crée-

```
def on_boBtnBas_clicked(self,widget):
      sel = self.treeview.get_selection()
      (modele,lignes) = sel.get_selected_rows()
      for chemin1 in lignes:
          chemin2 = self.NombreDeLignes-1
      iter1=modele.get_iter(chemin1)
      iter2 = modele.get_iter(chemin2)
      modele.move_after(iter1,iter2)
```

```
def on_boBtnMonter_clicked(self,widget):
      sel = self.treeview.get_selection()
      (modele,lignes) = sel.get_selected_rows()
      for chemin1 in lignes:
          chemin2 = (chemin1[0]-1,)
      if chemin2[0] >= 0:
          iter1=modele.get_iter(chemin1)
          iter2 = modele.get_iter(chemin2)
          modele.swap(iter1,iter2)
```

```
def on_boBtnDescendre_clicked(self,widget):
      sel = self.treeview.get_selection()
      (modele,lignes) = sel.get_selected_rows()
      for chemin1 in lignes:
          chemin2 = (chemin1[0]+1,)
      iter1=modele.get_iter(chemin1)
      if chemin2[0] <= self.NombreDeLignes-1:
          iter2 = modele.get_iter(chemin2)
          modele.swap(iter1,iter2)
```

```
#EXTM3U
Adult Contemporary/Chris Rea/Collection/02 - On The Beach.mp3
Adult Contemporary/Chris Rea/Collection/07 - Fool (If You Think It's Over).mp3
Adult Contemporary/Chris Rea/Collection/11 - Looking For The Summer.mp3
```

rons une fonction pour gérer la lecture d'un fichier MP3 et renvoyer le nom de l'artiste, le titre de la chanson et sa longueur en secondes, qui sont les trois informations dont nous avons besoin pour la ligne des informations étendues. Placez cette fonction après la fonction APropos dans la classe CreateurListeDeLecture (page suivante, en haut à droite).

À nouveau, pour vous rafraîchir la mémoire, je vais parcourir le code. Tout d'abord nous effaçons les trois variables de retour pour qu'elles soient renvoyées vides si quelque chose se passe de travers. Ensuite on passe le nom du fichier MP3 que nous allons examiner. Puis on place les clés dans (vous l'avez deviné) un itérateur et on parcourt cet itérateur en cherchant les deux balises spécifiques. Ce sont TPE1 pour le nom de l'artiste et TIT2 pour le titre de la chanson. Si jamais la clé n'existe pas, on obtiendra une erreur, donc on entoure chaque appel avec une instruction try|except. Ensuite on va chercher la longueur de la chanson dans l'attribut audio.info.length et on retourne tout ça.

On va maintenant modifier la fonction SauvegarderListe pour qu'elle supporte la ligne d'informations éten-dues. Tant que nous y sommes, véri-fions si le nom de fichier existe et, si c'est le cas, prévenons l'utilisateur et sortons de la routine. Aussi, pour rendre les choses un peu plus faciles pour l'utilisateur et, puisqu'on ne supporte aucun autre type de fichier, ajoutons automatiquement l'extension .m3u au chemin et au nom de fichier si elle n'y est pas déjà. Commençons par ajouter une ligne «_import os.path » au début du code entre les import de sys et de mutagen (à droite).

```
def RecupererInfoMP3(self,nomFichier):
      artiste = ''
      titre = ''
      longueurChanson = 0
      audio = MP3(nomFichier)
      cles = audio.keys()
      for cle in cles:
          try:
              if cle == "TPE1":          # Artiste
                  artiste = audio.get(cle)
          except:
              artiste = ''
          try:
              if cle == "TIT2":          # Titre de la chanson
                  titre = audio.get(cle)
          except:
              titre = ''
          longueurChanson = audio.info.length    # longueur de la
chanson
      return (artiste,titre,longueurChanson)
```

```
import os.path
```

Ensuite continuez et commentez la fonction SauveListeLecture actuelle et nous allons la remplacer.

```
def SavePlaylist(self):
      fp = self.txtPath.get_text()     # Get the file path from the text box
      fn = self.txtFilename.get_text() # Get the filename from the text box
      if fp == "":  # IF filepath is blank...
          self.MessageBox("error","Please provide a filepath for the playlist.")
      elif fn == "": # IF filename is blank...
          self.MessageBox("error","Please provide a filename for the playlist file.")
      else:  # Otherwise
```

Jusqu'ici la routine est la même. Voici où les changements commencent.

```
            debutExt = nf.rfind(".") # cherche le debut de l'extension
      if debutExt == -1:
          nf += '.m3u' # ajouter une extension s'il n'y en a pas
          self.txtNomFichier.set_text(nf) # remplace le nom de fichier dans le champ de texte
```

Tout comme pour la fonction AjouterFichiers, nous utiliserons la méthode rfind pour trouver la position du dernier point (« . ») dans le nom du fichier nf. S'il n'y en a pas, la valeur renvoyée sera -1. Donc nous vérifions si la valeur retournée est -1 et si c'est le cas on ajoute l'extension et on replace le nom du fichier dans le champ de texte pour être sympa.

```
if os.path.exists(fp + "/" + fn):


self.MessageBox(""erreur","Le fichier existe deja. Choisissez un autre nom.")
```

Ensuite on veut entourer le reste de la fonction dans une clause IF|ELSE (en haut à droite) pour que, si le fichier existe déjà, on puisse simplement sortir de la routine. On utilise os.path.exists(nom du fichier) pour cette vérification.

Le reste du code sert principalement à sauvegarder comme précédemment, mais regardons-le quand même.

```
else:
          fic = open(cf + "/" + nf,"w")  # ouvre le fichier
          fic.writelines('#EXTM3U\n')    # affiche l'en-tete M3U
          for ligne in self.listeLecture:
             nomFic = "%s/%s.%s" % (ligne[2],ligne[0],ligne[1])
             artiste,titre,longueurChanson = self.RecupererInfoMP3(nomFic)
             if longueurChanson > 0 and (artiste != '' and titre != ''):
                fic.writelines("#EXTINF:%d,%s - %s\n" %
(longueurChanson,artiste,titre))
             fic.writelines("%s\n" % nomFic)
          fic.close  # referme le fichier
          self.MessageBox("info","Liste de lecture sauvegardee !")
```

La ligne 2 ouvre le fichier dans lequel nous allons écrire. La ligne 3 y place l'en-tête M3U. La ligne 4 règle un parcours à travers la liste de lecture ListStore. La ligne 5 crée le nom du fichier à partir des trois colonnes de ListStore. La ligne 6 appelle RecupererInfoMP3 et stocke les valeurs renvoyées dans des variables. La ligne 7 vérifie ensuite si nous avons des valeurs dans toutes ces variables. Si c'est le cas, on écrit la ligne d'informations étendues à la ligne 8, sinon on n'essaie pas. La ligne 9 écrit la ligne du nom du fichier comme précédemment. La ligne 10 ferme gentiment le fichier et la ligne 11 affiche un message à l'utilisateur indiquant que tout est terminé.

```
def SetupBullesAide(self):
      self.boBtnAjouter.set_tooltip_text("Ajoute un ou des fichier(s) a la liste de lecture.")
      self.boBtnAPropos.set_tooltip_text("Affiche les informations sur le programme.")
      self.boBtnSupprimer.set_tooltip_text("Supprime l'entree selectionnee de la liste.")
      self.boBtnEffacer.set_tooltip_text("Supprime toutes les entrees de la liste.")
      self.boBtnQuitter.set_tooltip_text("Quitte le programme.")
      self.boBtnHaut.set_tooltip_text("Deplace l'entree selectionne tout en haut de la liste.")
      self.boBtnMonter.set_tooltip_text("Remonte l'entree selectionnee dans la liste.")
      self.boBtnDescendre.set_tooltip_text("Descend l'entree selectionne dans la liste.")
      self.boBtnBas.set_tooltip_text("Deplace l'entree selectionnee tout en bas de la liste.")
      self.btnNomRepertoire.set_tooltip_text("Choisis le repertoire de sauvegarde de la liste.")
      self.btnSauvegarderListe.set_tooltip_text("Sauvegarde la liste.")
      self.txtNomFichier.set_tooltip_text("Entrez ici le nom du fichier a sauvegarder. L'extension .m3u sera ajoutee
pour vous si vous l'oubliez.")
```

Allez, sauvegardez votre code et essayez-le.

À ce stade, la seule chose qu'on pourrait encore ajouter serait des bulles d'aide lorsque l'utilisateur survole nos contrôles avec sa souris. Cela y ajoute un air professionnel (ci-dessous). Créons maintenant une fonction pour faire cela.

Nous utilisons le widget references que nous avons réglé plus haut, puis on règle le texte pour la bulle d'aide avec (vous l'aurez deviné) l'attribut set_tooltip_text. Ensuite on doit ajou-ter l'appel à la routine. Retournez dans la routine __init__, après la ligne self.ReferencesWidgets, ajoutez :

```
self.SetupBullesAide()
```

Enfin et surtout (!), on veut placer notre logo dans la boîte APropos. Comme tout le reste ici, il y a un attribut pour faire cela. Ajoutez la ligne suivante à la routine APropos :

```
apropos.set_logo(gtk.gdk.pix
buf_new_from_file("logo.png"
))
```

Et voilà. Vous avez maintenant une application complète, fonctionnelle et jolie, qui fait un travail merveilleux de création de liste de lecture pour vos fichiers de musique.

Le code complet, incluant le fi-chier glade que nous avons créé le mois dernier, est disponible ici : http://pastebin.com/ZfZ69zVJ

Profitez des nouveaux talents que vous vous êtes découverts, jusqu'à la prochaine fois.

TUTORIEL

Programmer en Python - Partie 24

Greg Walters

Waouh ! Il est difficile de croire que ceci est déjà le 24e numéro. Cela fait deux ans que nous apprenons le Python ! Vous avez parcouru un très long chemin.

Cette fois-ci, nous allons traiter deux sujets. Le premier est l'impression sur une imprimante, le second est la création de fichiers RTF (Rich Text Format, ou Format de Texte Riche) comme sortie.

Impression générique sous Linux

Commençons donc avec l'impression sur une imprimante. L'idée de parler de cela provient d'un courriel envoyé par Gord Campbell. Il est réellement facile de faire la plupart des impressions depuis Linux ; plus facile qu'avec cet autre système d'exploitation qui commence par « WIN » - et dont je ne parlerai pas.

Tout est plutôt facile tant que vous ne souhaitez imprimer que du texte simple, sans gras, italique, changements de polices, etc. Voici une application simple qui permet d'imprimer directement sur votre imprimante :

```
import os

pr = os.popen('lpr','w')

pr.write('Test imprimante
depuis linux via python\n')

pr.write('Impression
terminee\n')

pr.close()
```

C'est assez facile à comprendre si vous élargissez un peu votre esprit. Dans le code ci-dessus, « lpr » est le spooler d'impression. Le seul prérequis est que nous ayons déjà configuré « lpd » et qu'il fonctionne. C'est très probablement déjà fait pour vous si vous utilisez une imprimante sous Ubuntu. « lpd » est généralement considéré comme un « filtre magique » qui permet de convertir automatiquement différents types de documents en quelque chose que l'imprimante peut comprendre. Nous allons imprimer sur le périphérique/objet « lpr ». Pensez-y comme à un simple fichier. Nous ouvrons le fichier ; nous devons importer « os ». Puis à la ligne 2, nous avons ouvert « lpr » avec un accès en écriture, en l'assignant à la variable objet « pr ». Nous procédons alors à une écriture « pr.write » avec tout ce que nous voulons imprimer. Enfin (ligne 5), nous fermons le fichier ce qui va envoyer les données vers l'imprimante.

Nous pouvons également créer un fichier texte puis l'envoyer à l'imprimante comme ceci…

```
import os

filename = 'fichier.bidon'

os.system('lpr %s' %
filename)
```

Dans ce cas, nous utilisons toujours l'objet lpr mais avec la commande « os.system » qui sert simplement à envoyer à Linux une commande comme si on l'avait saisie depuis un terminal.

Je vous laisserai vous amuser un peu avec cela.

> **Waouh ! Il est difficile de croire que ceci est déjà le 24e numéro. Cela fait deux ans que nous apprenons le Python !**

PyRTF

Maintenant occupons-nous des fichiers RTF. Le format RTF (c'est comme quand on dit le numéro PIN puisque PIN signifie Numéro d'Identification Personnel et que ça revient à dire le Numéro Numéro d'Identification Personnel [Ndt : en français on n'a pas ce problème de redondance puisqu'on parle de code PIN] : ça dépend du département Département des Redondances, non ?) a été créé à l'origine par Microsoft en 1987 et sa syntaxe s'est inspirée du langage de composition de texte TeX. PyRTF est une merveilleuse bibliothèque qui facilite la création de fichiers RTF. Cela nécessite de réfléchir en amont à ce à quoi le fichier doit ressembler, mais le résultat en vaut vraiment la peine.

Tout d'abord il faut télécharger et installer le paquet pyRTF. Allez sur http://pyrtf.sourceforge. et récupérez le paquet PyRTF-0.45.tar.gz. Sauvegardez-le quelque part et utilisez le gestionnaire d'archives pour le décompresser. Puis ouvrez un terminal et déplacez-vous à l'endroit où vous l'avez décompressé. Tout d'abord il faut installer le paquet, avec la commande « sudo python setup.py install ». Remarquez qu'il y a un répertoire d'exemples, qui contient de bonnes informations pour faire des choses un peu compliquées.

Nous y voilà. Commençons comme d'habitude en créant le canevas de notre programme que vous pouvez voir en haut à droite. Avant d'aller plus loin, parlons de ce qui se passe. La ligne 2 importe la bibliothèque pyRTF. Remarquez que nous utilisons un format d'importation différent des autres fois : cette fois-ci nous importons tout ce qui se trouve dans la biblothèque. Notre routine principale s'appelle FabriqueExemple et ne fait rien pour le moment. La routine OuvreFichier crée un fichier avec pour nom celui passé en argument, lui ajoute l'extension .rtf, le place en mode écriture et retourne un pointeur sur ce fichier.

Nous avons déjà parlé de la routine __name__ précédemment, mais pour vous rafraîchir la mémoire je vous rappelle que si nous exécutons le programme en mode autonome la variable interne __name__ est réglée à « __main__ » ; par contre, si on l'appelle comme « import » depuis un autre programme, cette portion de code sera ignorée.

Nous créons là une instance de l'objet Renderer, appelons la routine FabriqueExemple et récupérons l'objet retourné docu. Puis nous écrivons le fichier (docu) en utilisant la routine OuvreFichier.

Passons maintenant au contenu de la routine principale FabriqueExemple. Remplacez l'instruction pass par le code ci-dessous.

```
docu = Document()
ss   = doc.StyleSheet
section = Section()
docu.Sections.append(section)

p = Paragraph(ss.ParagraphStyles.Normal)
 p.append('Voici notre premier exemple de création de fichier RTF. '
       'Ce premier paragraphe est dans le style prédéfini appelé normal '
       'et tous les paragraphes suivants utiliseront ce style sauf si on le change.')
section.append(p)

return docu
```

Regardons ce que nous avons fait. La première ligne crée une instance de document. Puis on crée une instance de feuille de style. Ensuite nous créons une instance de l'objet section et on l'ajoute au document. Imaginez une section comme un chapitre dans un livre. Ensuite nous créons un paragraphe en utilisant le style Normal. L'auteur de pyRTF a préréglé ce style avec une police Arial en 11 points. Ensuite on écrit le texte qu'on veut dans ce paragraphe, on l'ajoute à la section et on retourne notre document docu.

C'est vraiment facile. Encore une fois, vous devez réfléchir soigneusement en amont à la sortie désirée, mais ce n'est pas très compliqué.
Sauvegardez ce programme en tant que « rtftesta.py » et exécutez-le. Enfin, utilisez OpenOffice (ou LibreOffice) pour ouvrir le fichier et l'examiner.

Maintenant faisons quelques modifications sympathiques. Tout d'abord, ajoutons un en-tête. Là encore l'auteur de pyRTF nous fournit un style prédéfini appelé Header1, que nous allons utiliser pour notre en-tête. Ajoutez ce qui suit entre les lignes docu.Sections.append et p = Paragraph.

```
p =
Paragraph(ss.ParagraphStyles.
Heading1)

p.append('Exemple d'en-tete')

section.append(p)
```

Modifiez le nom du fichier en «_rtftestb » ; cela devrait donner ceci :

```
DR.Write(docu,
OuvreFichier('rtftestb'))
```

Sauvegardez-le sous le nom rtftestb.py et exécutez-le. Maintenant nous avons un en-tête. Je suis sûr que votre esprit est en train de se demander tout ce qu'on peut faire encore. Voici une liste des styles prédéfinis que l'auteur nous fournit.

Normal, Normal Short, Heading 1, Heading 2, Normal Numbered, Normal Numbered 2. Il y a également un style List que je vous laisserai découvrir. Si vous voulez en voir davantage, sur ça et sur d'autres sujets, les styles sont définis dans le fichier Elements.py que vous avez installé tout à l'heure.

> **Voyons maintenant comment modifier les polices, leur taille et leurs attributs (gras, italique, etc.) à la volée.**

Ces styles prédéfinis sont utiles pour beaucoup de choses, mais on peut avoir besoin d'en créer d'autres. Voyons maintenant comment modifier les polices, leur taille et leurs attributs (gras, italique, etc.) à la volée. Après notre paragraphe, et avant de retourner l'objet document, insérez le code ci-dessus à droite et modifiez le nom du fichier de sortie en rtftestc.

```
  p = Paragraph(ss.ParagraphStyles.Normal)
p.append( 'Il est aussi possible de passer outre les elements d''un style. ',
        'Par exemple vous pouvez modifier seulement la ',
        TEXT(' taille de la police a 24 points', size=48),
        ' ou',
        TEXT(' son type a Impact', font=ss.Fonts.Impact),
        ' ou meme d''autres attributs comme',
        TEXT(' LA GRAISSE',bold=True),
        TEXT(' ou l''italique',italic=True),
        TEXT(' ou LES DEUX',bold=True,italic=True),
        '.' )
  section.append(p)
```

Sauvegardez le fichier sous le nom rtftestc.py et exécutez-le. La nouvelle portion du document devrait ressembler à ceci…

Il est également possible de passer outre les éléments d'un style. Par exemple vous pouvez modifier seulement la taille de la police à 24 points, ou son type à Impact ou même modifier d'autres attributs comme la graisse ou l'italique ou les deux.

Bon, qu'avons-nous fait ? La ligne 1 crée un nouveau paragraphe. On commence comme auparavant à l'ajouter au texte. Regardez la ligne 4 (TEXT(' taille de la police a 24 points', size=48),) : en utilisant le qualificatif TEXT on indique à pyRTF qu'il faut faire quelque chose de différent au milieu de la phrase, dans ce cas on modifie la taille de la police (Arial) à 24 points, en précisant à la suite la commande « size = ». Mais attendez une minute : on indique 48 comme taille alors qu'on veut écrire en 24 points ; et la sortie est réellement en 24 points. Que se passe-t-il ici ? Eh bien, la commande de taille est en demi-points ; ainsi si on veut écrire en police 8 points, on doit utiliser « size = 16 ». Vous comprenez ?

Ensuite, on continue le texte et on modifie la police avec la commande «_font = ». Cette fois encore, tout ce qui est dans l'instruction en ligne TEXT entre les guillemets sera affecté, mais pas le reste.

Bien. Si vous avez compris tout cela, que peut-on faire d'autre ?

On peut aussi régler la couleur du texte avec l'instruction en ligne TEXT de cette façon :

```
p = Paragraph()

p.append('Voici un nouveau
paragraphe avec le mot ',


TEXT('ROUGE',colour=ss.Colou
rs.Red),

    ' ecrit en rouge.')

section.append(p)
```

Remarquez que nous n'avons pas eu à repréciser que le style de paragraphe est Normal, puisqu'il ne change pas tant qu'on ne lui dit pas. Remarquez également que si vous habitez aux États-Unis vous devez utiliser la bonne orthographe pour « Colours_» [Ndt : les Américains utilisent souvent l'orthographe « impropre » Color].

Voici les couleurs prédéfinies : Black,

Blue, Turquoise, Green, Pink, Red, Yellow, White, BlueDark, Teal, GreenDark, Violet, RedDark, YellowDark, GreyDark et Grey.

Et voici une liste de toutes les polices prédéfinies (ce sont les notations pour les utiliser) :

Arial, ArialBlack, ArialNarrow, BitstreamVeraSans, BitstreamVeraSerif, BookAntiqua, BookmanOldStyle, Castellar, CenturyGothic, ComicSansMS, CourierNew, FranklinGothicMedium, Garamond, Georgia, Haettenschweiler, Impact, LucidaConsole, LucidaSansUnicode, MicrosoftSansSerif, PalatinoLinotype, MonotypeCorsiva, Papyrus, Sylfaen, Symbol, Tahoma, TimesNewRoman, TrebuchetMS et Verdana.

Maintenant vous devez penser que tout cela est bien joli, mais comment peut-on créer ses propres styles ? C'est assez simple. Retournez en haut de notre fichier et ajoutez le code qui suit avant la ligne d'en-tête.

```
result = doc.StyleSheet

NormalText =
TextStyle(TextPropertySet
(result.Fonts.CourierNew,16)
)

ps2 =
ParagraphStyle('Courier',
NormalText.Copy())

result.ParagraphStyles.appen
d(ps2)
```

Avant d'écrire le code pour l'utiliser, regardons ce que nous avons fait. Nous créons une nouvelle instance de feuille de style nommée result. À la deuxième ligne nous réglons la police à CourierNew en 8 points puis « enregistrons » le style comme Courier. Souvenez-vous que nous devons indiquer 16 comme taille puisque ce sont des demi-points.

Maintenant, ajoutons un nouveau paragraphe en utilisant le style Courier, avant la ligne return en bas de la routine.

```
  p = Paragraph(ss.ParagraphStyles.Courier)
p.append('Now we are using the Courier style at 8 points. '
         'All subsequent paragraphs will use this style automatically. '
         'This saves typing and is the default behaviour for RTF documents.',LINE)
section.append(p)
p = Paragraph()
p.append('Also notice that there is a blank line between the previous paragraph ',
         'and this one.  That is because of the "LINE" inline command.')

section.append(p)
```

Maintenant que vous avez un nouveau style, vous pouvez l'utiliser quand vous le souhaitez. Vous pouvez utiliser n'importe quelle police de la liste ci-dessus et créer vos propres styles. Recopiez simplement le code du style et remplacez les informations de police et de taille comme vous le voulez. On peut aussi faire cela :

```
NormalText =
TextStyle(TextPropertySet
(result.Fonts.Arial,22,bold=
True,colour=ss.Colours.Red))

ps2 =
ParagraphStyle('ArialGrasRouge'
,NormalText.Copy())

result.ParagraphStyles.append
(ps2)
```

Et ajouter le code suivant :

```
p =
Paragraph(ss.ParagraphStyles.
ArialGrasRouge)

p.append(LINE,'Et
maintenant on
utilise le style
ArialGrasRouge.',LINE)

section.append(p)
```

pour afficher en style ArialGrasRouge.

Tableaux

Souvent, la seule manière de présenter correctement des données dans un document est d'utiliser un tableau. Faire des tableaux dans un texte est plutôt difficile, mais PARFOIS c'est plutôt facile avec pyRTF. J'expliquerai cela plus tard dans cet article.

Regardons un tableau standard (ci-dessous) dans OpenOffice/LibreOffice. Cela ressemble à une feuille de calcul, où tout est placé dans des colonnes.

Des lignes horizontales, et des colonnes verticales. Un concept simple.

Commençons une nouvelle application nommée rtfTableau-a.py. Démarrons avec notre code standard (page suivante) et construisons à partir de là.

Pas besoin d'explications ici puisque c'est à peu près le même code que nous avons utilisé précédemment. Maintenant, écrivons la routine ExempleTableau. J'utilise en partie l'exemple fourni par l'auteur de pyRTF. Remplacez l'instruction pass dans la routine par le code suivant :

```
docu = Document()

ss = docu.StyleSheet

section = Section()

docu.Sections.append(section)
```

Cette partie est la même que précédemment, passons à la suite.

```
tableau =
Table(TabPS.DEFAULT_WIDTH * 7,

      TabPS.DEFAULT_WIDTH * 3,

      TabPS.DEFAULT_WIDTH * 3)
```

Cette ligne (oui, il ne s'agit que d'une ligne, mais découpée pour plus de clarté) crée notre tableau basique. Nous créons un tableau à trois colonnes, la première contient 7 cellules, les deux suivantes en contiennent 3. Nous n'avons pas que des cellules uniques à notre disposition, car on pourra saisir les largeurs en « twips » [Ndt : ce sont des unités de mesure, utilisées en LibreOffice et pour le RTF, équivalentes à 1/1440 d'un pouce (2,54 cm). Cf http://en.wikipedia.org/wiki/Twip#In_computing.] Nous y reviendrons dans un moment.

```
c1 = Cell(Paragraph('ligne 1,
cellule 1'))

c2 = Cell(Paragraph('ligne 1,
cellule 2'))

c3 = Cell(Paragraph('ligne 1,
cellule 3'))

tableau.AddRow(c1,c2,c3)
```

Ici nous réglons les données qui vont dans chaque cellule de la première ligne.

```
c1 =
Cell(Paragraph(ss.ParagraphS
tyles.Heading2,'Style en-
tete 2'))

c2 =
Cell(Paragraph(ss.ParagraphS
tyles.Normal,'Retour au
style Normal'))

c3 = Cell(Paragraph('Encore
du style Normal'))

tableau.AddRow(c1,c2,c3)
```

```
#!/usr/bin/env python

from PyRTF import *

def ExempleTableau():
  pass

def OuvreFichier(nom):
  return file('%s.rtf' % nom, 'w')

if __name__ == '__main__':
  DR = Renderer()
  docu = ExempleTableau()
  DR.Write(docu, OuvreFichier('rtftable-a'))
  print "Fini"
```

Ce morceau de code règle les données pour la deuxième ligne. Remarquez que nous pouvons régler des styles différents pour une seule ou plusieurs cellules.

```
c1 =
Cell(Paragraph(ss.ParagraphS
tyles.Heading2,'Style en-
tete 2'))

c2 =
Cell(Paragraph(ss.ParagraphS
tyles.Normal,'Retour au
style Normal'))

c3 = Cell(Paragraph('Encore
du style Normal'))

tableau.AddRow(c1,c2,c3)
```

Ceci règle la dernière ligne.

```
section.append(tableau)

return docu
```

Ceci ajoute le tableau dans la section et retourne le document pour affichage.

Sauvegardez et exécutez l'application. Vous remarquerez que tout ressemble à ce que vous attendiez, mais qu'il n'y a pas de bordures pour le tableau. Ceci peut rendre les choses difficiles à lire : réglons ce problème. À nouveau, j'utilise en grande partie le code de l'exemple fourni par l'auteur de pyRTF.

Sauvegardez votre fichier sous le nom rtfTableau-b.py, puis effacez tout ce qui est entre « docu.Sections.append(section) » et « return docu » dans la routine ExempleTableau, et remplacez-le par ce qui suit :

```
cote_fin  = BorderPS( width=20,
style=BorderPS.SINGLE )
cote_epais = BorderPS( width=80,
```

```
style=BorderPS.SINGLE )

bord_fin  =  FramePS( cote_fin,
cote_fin,   cote_fin,   cote_fin )
bord_epais = FramePS( cote_epais,
cote_epais, cote_epais, cote_epais )

bord_mixte = FramePS( cote_fin,
cote_epais, cote_fin, cote_epais )
```

Ici nous réglons les définitions des côtés et des bords pour les encadrements.

```
tableau = Table(
TabPS.DEFAULT_WIDTH * 3,
TabPS.DEFAULT_WIDTH * 3,
TabPS.DEFAULT_WIDTH * 3 )

c1 = Cell( Paragraph( 'L1C1' ),
bord_fin )

c2 = Cell( Paragraph( 'L1C2' ) )

c3 = Cell( Paragraph( 'L1C3' ),
bord_epais )

tableau.AddRow( c1, c2, c3 )
```

Dans la première ligne, les cellules de la colonne 1 (bord_fin) et de la colonne 3 (bord_epais) auront une bordure.

```
c1 = Cell( Paragraph( 'L2C1' ) )

c2 = Cell( Paragraph( 'L2C2' ) )

c3 = Cell( Paragraph( 'L2C3' ) )

tableau.AddRow( c1, c2, c3 )
```

Aucune des cases n'aura de bordure dans la ligne 2.

```
c1 = Cell( Paragraph( 'L3C1' ),
bord_mixte )

c2 = Cell( Paragraph( 'L3C2' ) )

c3 = Cell( Paragraph( 'L3C3' ),
bord_mixte )

tableau.AddRow( c1, c2, c3 )
```

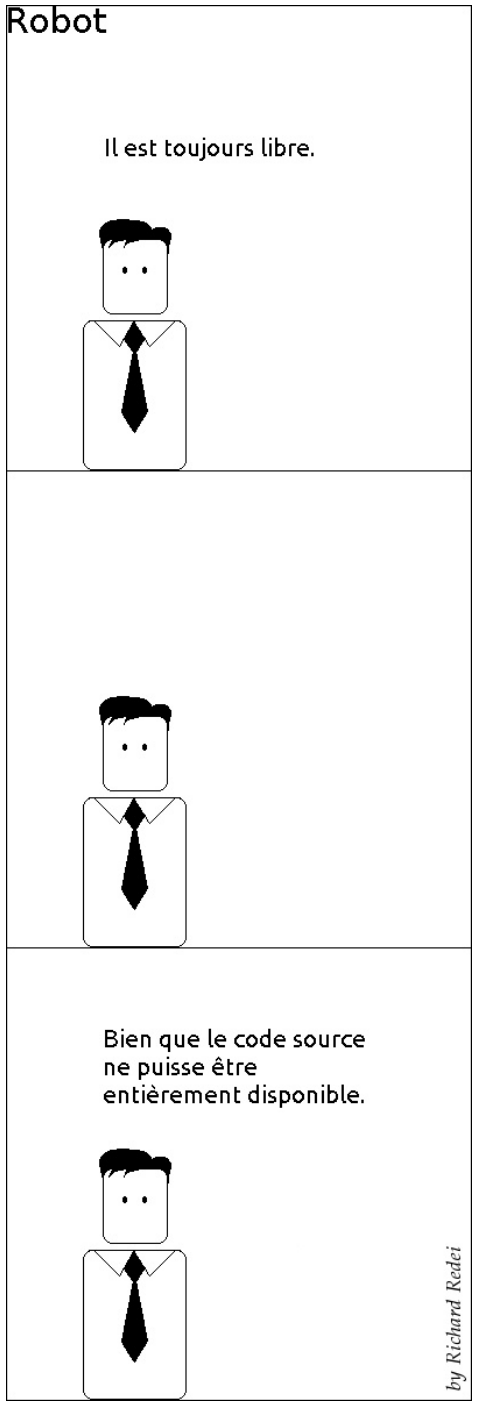
À nouveau, les cases des colonnes 1 et 3 auront une bordure mixte dans la troisième ligne.

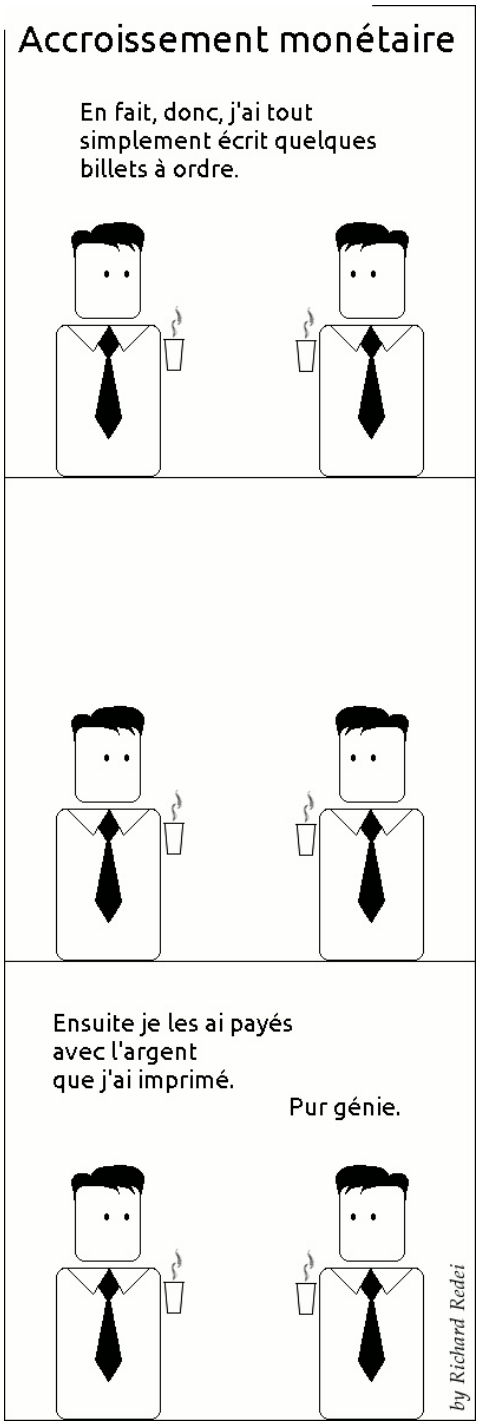
```
section.append( tableau )
```

Et voilà. Vous avez maintenant les bases pour créer des documents RTF avec du code.

À la prochaine fois !

Le code source est disponible sur pastebin comme d'habitude. La première partie est ici : http://pastebin.com/uRVrGjkV et contient le résumé de rtftest.py (a à e), la seconde partie rtfTableau.py (a et b) est ici : http://pastebin.com/L8DGU7Lz.





TUTORIEL

Programmer en Python - Partie 25

Greg Walters

Un certain nombre d'entre vous ont commenté les articles de programmation graphique et dit combien vous les avez appréciés. En réponse à cela, nous allons commencer à jeter un œil à un autre outil d'interfaces graphiques appelé Tkinter. Ceci est la façon « officielle » de faire de la programmation graphique en Python. Tkinter existe depuis longtemps et a une assez mauvaise réputation pour son côté « démodé ». Ceci a changé récemment, alors j'ai pensé que nous pourrions nous battre contre ce mauvais processus de réflexion.

N.B. : Tout le code présenté ici est pour Python 2.x seulement. Dans un prochain article, nous allons discuter de la façon d'utiliser Tkinter avec Python 3.x. Si vous DEVEZ utiliser Python 3.x, changez les déclarations d'importation en « from tkinter import * ».

Un peu d'histoire et un peu de contexte

Tkinter est l'abbréviation de « Tk interface ». Tk est un langage de programmation à lui tout seul, et le module Tkinter nous permet d'utiliser les fonc-tions de l'interface graphique de ce langage. Il y a un certain nombre de widgets qui viennent nativement avec le module Tkinter. Parmi eux, on trouve des conteneurs de haut niveau (des fenêtres principales), des boutons, des étiquettes, des cadres, des zones de saisie de texte, des cases à cocher, des boutons radio, des canevas, des entrées de texte multilignes, et bien plus encore. Il y a aussi de nombreux modules qui ajoutent des fonctionnalités par dessus Tkinter. Ce mois-ci, nous allons nous concentrer sur quatre widgets. Un conteneur de haut niveau (à partir d'ici je vais essentiellement l'appeler la fenêtre racine), un cadre, des étiquettes et des boutons. Dans le prochain article, nous verrons plus de widgets plus en profondeur.

Fondamentalement, nous avons le widget conteneur de haut niveau qui contient d'autres widgets. Il s'agit de la fenêtre racine ou principale. Dans cette fenêtre racine, nous plaçons les widgets que nous voulons utiliser dans notre programme. Chaque widget, à l'exception du conteneur racine principal, a un parent. Le parent n'est pas forcément la fenêtre racine ; ça peut être un autre widget. Nous verrons cela le mois prochain. Pour ce mois-ci, tous les widgets auront pour parent la fenêtre racine.

```
    COLUMNS - >
ROWS | 0,0 | 1,0 | 2,0 | 3,0 | 4,0 |
  |  | 0,1 | 1,1 | 2,1 | 3,1 | 4,1 |
     | 0,2 | 1,2 | 2,2 | 3,2 | 4,2 |
     | 0,3 | 1,3 | 2,3 | 3,3 | 4,3 |
```

Afin de placer et d'afficher les widgets enfants, nous devons utiliser ce qu'on appelle la « gestion de géométrie ». C'est la façon dont les choses se placent dans la fenêtre racine principale. La plupart des programmeurs utilisent un de ces trois types de gestion de géométrie : Packer, Grid, ou Gestion de la place. À mon humble avis, la méthode Packer est très maladroite. Je vous laisse l'explorer par vous-même. La méthode de gestion de la place permet un placement extrêmement précis des widgets, mais ça peut être compliqué. Nous en reparlerons dans un futur article. Cette fois-ci, nous allons nous concentrer sur la méthode de la grille.

Pensez à un tableur. Il y a des lignes et des colonnes. Les colonnes sont verticales, les lignes sont horizontales. Voici une représentation texte simple des adresses de cellule d'une grille simple de 5 colonnes sur 4 lignes (en haut à droite). Le parent possède la grille, les widgets vont dans les positions de la grille. Au premier regard, vous pourriez penser que cela est très limitatif. Toutefois, les widgets peuvent s'étendre sur plusieurs positions sur la grille, soit dans le sens des colonnes, soit dans celui des lignes, ou les deux à la fois.

Notre premier exemple

Notre premier exemple est SUPER simple (seulement quatre lignes), mais explicite.

```
from Tkinter import_*

racine = Tk()

bouton = Bouton(racine, text
= "Bonjour FullCircle").grid()

racine.mainloop()
```

Bon, qu'est-ce qui se passe ici ? La première ligne importe la bibliothèque Tkinter. Ensuite, on instancie l'objet Tk racine (Tk est une partie de Tkinter). Voici la ligne trois :

```
bouton = Button(racine, text
= "Bonjour FullCircle").grid()
```

Nous créons un bouton appelé bouton, définissons son parent à la fenêtre racine, réglons son texte à « Bonjour FullCircle » et le plaçons dans la grille. Enfin, nous appelons la boucle principale de la fenêtre. Ça paraît très simple quand on regarde le code, mais beaucoup de choses se passent dans les coulisses. Heureusement, nous n'avons pas besoin de comprendre tout cela pour l'instant.

Exécutez le programme et nous allons voir ce qui se passe. Sur ma machine, la fenêtre principale apparaît en bas à gauche de l'écran. Elle pourrait appa-raître ailleurs sur le vôtre. Cliquer sur le bouton ne fait rien. Réparons cela dans notre prochain exemple.

Notre deuxième exemple

Cette fois, nous allons créer une classe appelée App. Ce sera la classe qui détient effectivement notre fenêtre.

```
class App:
def __init__(self, principale):
        cadre = Frame(principale)
        self.lblTexte = Label(cadre, text = "Voici un widget label")
        self.btnQuitter = Button(cadre, text="Quitter", fg="red", command=cadre.quit)
        self.btnBonjour = Button(cadre, text="Bonjour", command=self.DitUnTruc)
        cadre.grid(column = 0, row = 0)
        self.lblTexte.grid(column = 0, row = 0, columnspan = 2)
        self.btnBonjour.grid(column = 0, row = 1)
```

Commençons :

```
from Tkinter import *
```

C'est la déclaration d'importation pour la bibliothèque Tkinter.

Nous définissons notre classe, et dans la routine __init__, nous mettons en place nos widgets et les plaçons dans la grille.

La première ligne dans la routine __init__ crée un cadre qui sera le parent de tous nos autres widgets. Le parent de ce cadre est la fenêtre racine (widget de plus haut niveau). Ensuite, nous définissons un label et deux boutons. Regardons la ligne de création de l'étiquette.

```
self.lblTexte = Label(cadre,
text = "Ceci est un widget
label")
```

Nous créons le widget étiquette et l'appelons self.lblTexte. Il hérite de l'objet widget Label. Nous réglons son parent (le cadre) et définissons le texte à afficher (text = "Ceci est un widget label"). C'est aussi simple que cela. Bien sûr, nous pouvons faire beaucoup mieux, mais pour l'instant c'est tout ce dont nous avons besoin. Ensuite, nous mettons en place les deux boutons que nous allons utiliser :

```
self.btnQuitter =
Button(cadre, text="Quitter",
fg="red", command=cadre.quit)
self.btnBonjour =
Button(cadre, text="Bonjour",
command=self.DitUnTruc)
```

Nous nommons les widgets, fixons leur parent (cadre) et définissons le texte à afficher. Maintenant btnQuitter a un attribut marqué fg que nous avons réglé à « red ». Vous avez deviné que cela définit la couleur d'avant-plan ou la couleur du texte à la couleur rouge. Le dernier attribut sert à définir la commande que nous voulons utiliser lorsque l'utilisateur clique sur le bouton. Dans le cas de btnQuitter, c'est cadre.quit, qui termine le programme. C'est une fonction intégrée, donc nous n'avons pas besoin de la créer. Dans le cas de btnBonjour, c'est une routine appelée self.DitUnTruc. Nous devons créer celle-ci, mais auparavant nous avons encore quelque chose à faire.

Nous devons placer nos widgets dans la grille. Voici les lignes à nouveau :

```
cadre.grid(column = 0, row =
0)

self.lblTexte.grid(column =
0, row = 0, columnspan = 2)

self.btnBonjour.grid(column
= 0, row = 1)

self.btnQuitter.grid(column
= 1, row = 1)
```

Tout d'abord, nous attribuons une grille au cadre. Ensuite, nous réglons l'attribut de grille de chaque widget

selon l'endroit où nous voulons placer le widget. Notez la ligne « columnspan » pour l'étiquette (self.lblTexte). Cela indique que nous voulons que l'étiquette s'étende sur deux colonnes de la grille. Puisque nous avons seulement deux colonnes, il s'agit de toute la largeur de l'application. Maintenant nous pouvons créer notre fonction de rappel :

```
def DitUnTruc(self):

    print "Bonjour lecteur
du Magazine FullCircle !!"
```

Cela affiche simplement dans la fenêtre du terminal le message “Bonjour lecteur du Magazine FullCircle !!” Enfin, on instancie la classe Tk - notre classe App - et exécutons la boucle principale :

```
racine = Tk()

app = App(racine)

racine.mainloop()
```

Essayez le programme. Maintenant, il fait vraiment quelque chose. Mais là encore, la position de la fenêtre est très gênante. Corrigeons cela dans notre prochain exemple.

Notre troisième exemple

Enregistrez l'exemple précédent sous le nom exemple3.py. Tout est exac-tement pareil, sauf une seule ligne qui se trouve en bas de la routine principale. Je vais vous montrer ces lignes avec la nouvelle :

```
root = Tk()

root.geometry('150x75+550+150')

app = App(root)

root.mainloop()
```

Ceci force notre fenêtre initiale à une taille de 150 pixels de large sur 75 pixels de haut. Nous voulons aussi que le coin supérieur gauche de la fenêtre soit placé à une position horizontale de 550 pixels (depuis la droite) et à une position verticale de 150 pixels (depuis le haut). Comment suis-je arrivé à ces chiffres ? J'ai commencé avec des valeurs raisonnables et les ai peaufiné à partir de là. C'est un peu difficile de faire de cette façon, mais les résultats sont meilleurs que si on ne fait rien du tout.

Notre quatrième exemple - Une calculatrice simple

Maintenant, regardons quelque chose d'un peu plus compliqué. Cette fois, nous allons créer une calculatrice simple à 4 boutons, pour les 4 opérations : addition, soustraction, multiplication et division. À droite vous voyez à quoi elle ressemblera, sous forme de texte simple.

```
----------------
|              0 |
----------------
| 1 | 2 | 3 | + |
----------------
| 4 | 5 | 6 | - |
----------------
| 7 | 8 | 9 | * |
----------------
| - | 0 | . | / |
----------------
|       =        |
----------------
|     CLEAR      |
----------------
```

```
from Tkinter import *

def StartUp():
    global val, w, root
    root = Tk()
    root.title('Easy Calc')
    root.geometry('247x330+469+199')
    w = Calculator(root)
    root.mainloop()
```

Nous allons y plonger tout de suite et je vous expliquerai le code (au milieu à droite) au fur et à mesure.

À part la déclaration de la géométrie, ceci devrait être assez facile pour vous de comprendre maintenant (à gauche). Rappelez-vous, prenez des valeurs raisonnables, modifiez-les, puis continuez.

```
class Calculator():
    def __init__(self,root):
        master = Frame(root)
        self.CurrentValue = 0
        self.HolderValue = 0
        self.CurrentFunction = ''
        self.CurrentDisplay = StringVar()
        self.CurrentDisplay.set('0')
        self.DecimalNext = False
        self.DecimalCount = 0
        self.DefineWidgets(master)
        self.PlaceWidgets(master)
```

Nous commençons notre définition de la classe en mettant en place notre fonction __init__. Nous réglons trois variables comme suit :

• ValeurCourante - Contient la valeur actuelle qui a été entrée dans la calculatrice.

• ValeurAncienne - Contient la valeur

qui existait avant que l'utilisateur ne clique sur une touche de fonction.

• FonctionCourante - C'est tout simplement pour se souvenir quelle fonction est traitée.

Ensuite, nous définissons la variable AffichageCourant et l'attribuons à l'objet StringVar. C'est un objet spécial qui fait partie de la trousse Tkinter. Quel que soit le widget auquel vous l'attribuez, cela met automatiquement à jour la valeur dans le widget. Dans ce cas, nous allons l'utiliser pour contenir ce que nous voulons que le widget d'affichage label… euh… eh bien… affiche. Nous devons l'instancier avant de pouvoir l'assigner au widget. Ensuite, nous utilisons la fonction « set » fournie par Tkinter. Nous définissons ensuite une variable boléenne appelée PartieDecimale et une variable CompteDecimales, puis nous appelons la fonction DefinirWidgets qui crée tous les widgets et ensuite nous appelons la fonction PlacerWidgets, qui les place réellement dans la fenêtre racine.

```
def
DefinirWidgets(self,principale):

self.lblAffichage =
Label(principale,
anchor=E,relief =
SUNKEN,bg="white",
height=2,textvar
iable=self.Affic
hageCourant)
```

Bon, nous avons déjà défini un label auparavant. Cependant, cette fois, nous ajoutons un certain nombre d'autres attributs. Notez que nous n'utilisons pas l'attribut « text ». Ici, nous assignons l'étiquette au parent (la fenêtre principale), puis nous définissons l'ancre (ou, pour nos fins, la justification) pour le texte lorsqu'il est écrit. Dans ce cas, nous précisons à l'étiquette de justifier tout le texte à l'est, c'est-à-dire sur le côté droit du widget. Il existe un attribut de justification, mais il sert lorsqu'il y a plusieurs lignes de texte. L'attribut d'ancrage a les options suivantes : N, NE, E, SE, S, SW, W, NW et CENTER. La valeur par défaut est de centrer. Vous devriez penser à des points cardinaux. Dans des circonstances normales, les valeurs réellement utilisables sont E (à droite), W (à gauche), et CENTER (pour centrer).

Ensuite, nous réglons le relief, qui est le style visuel de l'étiquette. Les options autorisées sont FLAT (à plat), SUNKEN (en creux), RAISED (en relief), GROOVE (en strie) et RIDGE (en arête). La valeur par défaut est à plat si vous ne spécifiez rien. N'hésitez pas à essayer les autres combinaisons par vous-mêmes lorsque nous aurons fini. Ensuite, nous définissons le fond (bg) à blanc afin de le démarquer un peu du reste de la fenêtre. Nous fixons la hauteur à 2 (qui signifie deux lignes de texte de haut, et non pas 2 pixels) et enfin nous assignons la variable que nous venons de définir juste avant (self.AffichageCourant) à l'attribut textvariable. À chaque fois que la valeur de self.AffichageCourant changera, le label modifiera son texte pour correspondre automatiquement.

Ci-dessus, nous allons créer quelques-uns des boutons.

```
self.btn1 = Button(master, text = '1',width = 4,height=3)
self.btn1.bind('<ButtonRelease-1>', lambda e: self.funcNumButton(1))
self.btn2 = Button(master, text = '2',width = 4,height=3)
self.btn2.bind('<ButtonRelease-1>', lambda e: self.funcNumButton(2))
self.btn3 = Button(master, text = '3',width = 4,height=3)
self.btn3.bind('<ButtonRelease-1>', lambda e: self.funcNumButton(3))
self.btn4 = Button(master, text = '4',width = 4,height=3)
self.btn4.bind('<ButtonRelease-1>', lambda e: self.funcNumButton(4))
```

J'ai montré seulement 4 boutons ici. C'est parce que, comme vous pouvez le voir, le code est presque exactement le même. Encore une fois, nous avons créé des boutons plus tôt dans ce tutoriel, mais nous allons regarder de plus près ce que nous faisons ici.

Nous commençons par définir le parent (la fenêtre principale), le texte que nous voulons sur le bouton, et la largeur et la hauteur. Notez que la largeur est en caractères et la hauteur est en lignes de texte. Si vous vouliez un graphique dans le bouton, vous utiliseriez des pixels pour définir la hauteur et la largeur. Cela peut devenir un peu confus jusqu'à ce que vous le compreniez sans faille. Ensuite, nous réglons l'attribut « bind ». Quand nous avons fait des boutons dans les exemples précédents, nous avons utilisé l'attribut « command= » pour définir quelle fonction serait appelée lorsque l'utilisateur clique sur le bouton. Cette fois, nous utilisons l'attribut « .bind » [Ndt : « relier »]. C'est presque la

même chose, mais c'est un moyen plus facile de le faire et de transmettre des informations à la routine de rappel qui est statique. Notez que nous utilisons ici « <ButtonRelease-1> » comme l'élément déclencheur de la liaison. Dans ce cas, nous voulons nous assurer que l'appel de la fonction se fait seulement après que l'utilisateur clique ET relâche le bouton gauche de la souris. Enfin, nous définissons la fonction de rappel que nous voulons utiliser et ce que nous allons lui envoyer. Maintenant, ceux d'entre vous qui sont astucieux (ce qui est bien sûr votre cas à tous) noteront quelque chose de nouveau : L'appel « lambda e: ».

En Python, nous utilisons Lambda pour définir des fonctions anonymes qui apparaîtront à l'interpréteur comme des instructions valides. Cela nous permet de mettre plusieurs morceaux dans une seule ligne de code. Pensez-y comme à une mini-fonction. Dans ce cas, nous mettons en place le nom de la fonction de rappel et la valeur que nous voulons lui envoyer, ainsi que la balise événement (e:). Nous parlerons plus en détail de Lambda dans un article ultérieur. Pour l'instant, il suffit de suivre l'exemple.

Je vous ai donné les quatre premiers

```
        self.btnDash = Button(principale, text = '-',width = 4,height=3)
        self.btnDash.bind('<ButtonRelease-1>', lambda e: self.foncBoutonFonction('SIGNE'))
        self.btnDot = Button(principale, text = '.',width = 4,height=3)
        self.btnDot.bind('<ButtonRelease-1>', lambda e: self.foncBoutonFonction('Dec'))
```

Le bouton « btnDash » change le signe de la valeur affichée. 523 devient -523 et -523 devient 523. Le bouton btnDot saisit un point décimal. Ces exemples, ainsi que les suivants, utilisent la fonction foncBoutonFonction.

```
        self.btnPlus = Button(principale,text = '+', width = 4, height=3)
        self.btnPlus.bind('<ButtonRelease-1>', lambda e: self.foncBoutonFonction('Ajouter'))
        self.btnMinus = Button(principale,text = '-', width = 4, height=3)
        self.btnMinus.bind('<ButtonRelease-1>', lambda e: self.foncBoutonFonction('Soustraire'))
        self.btnStar = Button(principale,text = '*', width = 4, height=3)
        self.btnStar.bind('<ButtonRelease-1>', lambda e: self.foncBoutonFonction('Multiplier'))
        self.btnDiv = Button(principale,text = '/', width = 4, height=3)
        self.btnDiv.bind('<ButtonRelease-1>', lambda e: self.foncBoutonFonction('Diviser'))
        self.btnEqual = Button(principale, text = '=')
        self.btnEqual.bind('<ButtonRelease-1>', lambda e: self.foncBoutonFonction('Egal'))
```

Voici les quatre boutons pour les fonctions mathématiques.

```
        self.btnClear = Button(principale, text = 'EFFACER')
        self.btnClear.bind('<ButtonRelease-1>', lambda e: self.foncEffacer())
```

Enfin, voici le bouton Effacer. Il efface bien sûr les variables et l'affichage. Maintenant nous plaçons les widgets avec la routine PlacerWidgets. D'abord nous initialisons la grille, puis nous plaçons les widgets dedans. Voici la première partie de la routine :

```
    def PlacerWidgets(self,principale):
        principale.grid(column=0,row=0)
        self.lblAffichage.grid(column=0,row=0,columnspan = 4,sticky=EW)
        self.btn1.grid(column = 0, row = 1)
        self.btn2.grid(column = 1, row = 1)
        self.btn3.grid(column = 2, row = 1)
        self.btn4.grid(column = 0, row = 2)
        self.btn5.grid(column = 1, row = 2)
        self.btn6.grid(column = 2, row = 2)
        self.btn7.grid(column = 0, row = 3)
        self.btn8.grid(column = 1, row = 3)
        self.btn9.grid(column = 2, row = 3)
        self.btn0.grid(column = 1, row = 4)
```

boutons. Copiez et collez le code ci-dessus pour les boutons de 5 à 9 et pour le bouton 0. Ils sont tous identiques, à l'exception du nom du bouton et de la valeur que nous envoyons au rappel. Les prochaines étapes sont indiquées à droite.

La seule chose dont nous n'avons pas parlé pour l'instant, ce sont les attributs « columnspan » et « sticky ». Comme je l'ai mentionné auparavant, un widget peut s'étendre sur plus d'une colonne ou une ligne. Dans ce cas, nous « étirons » le widget étiquette sur les quatre colonnes. C'est ce que fait l'attribut « columnspan ». Il existe également un attribut « rowspan ». L'attribut « sticky » [Ndt : « collant »] indique au widget où aligner ses bords. Pensez-y comme la manière dont le widget se remplit au sein de la grille. En haut à gauche vous voyez le reste de nos boutons.

Avant d'aller plus loin nous allons jeter un œil à la façon dont les choses vont fonctionner quand l'utilisateur appuiera sur les boutons.

Disons que l'utilisateur veut saisir 563 + 127 et obtenir la réponse. Il appuiera ou cliquera (logiquement) sur 5, puis 6, puis 3, puis le « + », puis 1, puis 2, puis 7, puis le bouton « = ».

```
self.btnDash.grid(column = 0, row = 4)
self.btnDot.grid(column = 2, row = 4)
self.btnPlus.grid(column = 3,row = 1)
self.btnMinus.grid(column = 3, row = 2)
self.btnStar.grid(column = 3, row = 3)
self.btnDiv.grid(column=3, row = 4)
self.btnEqual.grid(column=0,row=5,columnspan = 4,sticky=NSEW)
self.btnClear.grid(column=0,row=6,columnspan = 4, sticky = NSEW)
```

```
def funcNumButton(self,val):
    if self.DecimalNext == True:
        self.DecimalCount += 1
        self.CurrentValue = self.CurrentValue + (val * (10**-self.DecimalCount))
    else:
        self.CurrentValue = (self.CurrentValue * 10) + val
    self.DisplayIt()
```

Comment pouvons-nous gérer cela dans le code ? Nous avons déjà réglé les rappels pour les touches numériques à la fonction foncBoutonNumerique. Il y a deux façons de gérer cela. Nous pouvons conserver les informations saisies comme une chaîne et puis la convertir en nombre quand nous avons besoin, ou bien nous pouvons le garder comme un nombre tout le temps. Nous allons utiliser cette dernière méthode. Pour ce faire, nous allons conserver la valeur qui est déjà là (0 quand nous commencerons) dans une variable appelée “self.ValeurCourante”, puis quand un nouveau chiffre arrive, nous prenons la variable, la multiplions par 10 et ajoutons la nouvelle valeur. Ainsi, lorsque l'utilisateur entre 5, 6 et 3, nous faisons les choses suivantes :

```
L'utilisateur clique 5 : 0 *
10 + 5 (5)

L'utilisateur clique 6 : 5 *
10 + 6 (56)

L'utilisateur clique 3 : 56 *
10 + 3 (563)
```

Bien sûr, nous devons ensuite afficher la variable « self.ValeurCourante » dans l'étiquette.

Ensuite, l'utilisateur clique sur le bouton « + ». Nous prenons la valeur de « self.ValeurCourante » et la plaçons dans la variable « self.ValeurAncienne » et réinitialisons “self.ValeurCourante” à 0. Nous devons ensuite répéter le pro-cessus pour les clics sur 1, 2 et 7. Lorsque l'utilisateur clique sur la touche « = », nous devons ensuite ajouter les valeurs de « self.ValeurCourante » et « self.Va-leurAncienne », les afficher, puis effacer les deux variables pour continuer.

Ci-dessus, voici le code pour commencer à définir nos fonctions de rappel.

La routine « foncBoutonNumerique » reçoit la valeur que nous lui passons en appuyant sur un bouton. La seule chose qui diffère de l'exemple ci-dessus est lorsque l'utilisateur appuie sur le bouton de décimale (« . »). Ci-dessous, vous verrez que nous utilisons une variable booléenne pour retenir le fait qu'il a déjà appuyé sur

le bouton décimal, et, au prochain clic, on s'en occupe. D'où la ligne « if self.PartieDecimale == True: ». Nous allons procéder pas à pas.

L'utilisateur clique sur 3, puis 2, puis le point décimal, puis 4 pour créer « 32.4 ». Nous traitons les clics sur 3 et 2 grâce à la routine « foncBoutonNumerique ». Nous vérifions pour voir si self.PartieDecimale est vrai (ce qu'il n'est pas tant que l'utilisateur n'a pas cliqué sur le bouton « . »). Sinon, nous multiplions simplement la valeur de self.ValeurCourante par 10 et ajoutons la nouvelle valeur. Lorsque l'utilisateur clique sur le « . », la fonction de rappel « foncBoutonFonction » est appelée avec la valeur « Dec ». Tout ce que nous faisons est de régler la variable booléenne « self.PartieDecimale » à vrai (True). Lorsque l'utilisateur clique sur le 4, nous allons tester la valeur de « self.PartieDecimale » et, puisqu'elle est vraie, nous faisons un peu de magie. Premièrement, on incrémente la variable self.CompteDecimales, qui nous indique le nombre de décimales avec lequel nous travaillons. Nous prenons ensuite la nouvelle valeur entrante, la multiplions par (10**</ nowiki>-self.CompteDecimales). En

```
    def foncBoutonFonction(self,fonction):
        if fonction == 'Dec':
            self.PartieDecimale = True
        else:
            self.PartieDecimale = False
            self.CompteDecimales = 0
            if fonction == 'SIGNE':
                self.ValeurCourante *= -1
                self.Rafraichir()4
```

La fonction ABS fournit simplement la valeur actuelle et la multiplie par -1.

```
            elif fonction == 'Ajouter':
                self.ValeurAncienne = self.ValeurCourante
                self.ValeurCourante = 0
                self.FonctionCourante = 'Ajouter'
```

La fonction Add copie “self.CurrentValue” dans “self.HolderValue”, nettoie “self.CurrentValue”, and sets the “self.CurrentFunction” to “Add”. Les fonctions Soustraire, Multiplier et Diviser font la même chose avec the proper keyword being set in “self.CurrentFunction”.

```
            elif fonction == 'Soustraire':
                self.ValeurAncienne = self.ValeurCourante
                self.ValeurCourante = 0
                self.FonctionCourante = 'Soustraire'
            elif fonction == 'Multiplier':
                self.ValeurAncienne = self.ValeurCourante
                self.ValeurCourante = 0
                self.FonctionCourante = 'Multiplier'
            elif fonction == 'Diviser':
                self.ValeurAncienne = self.ValeurCourante
                self.ValeurCourante = 0
                self.FonctionCourante = 'Diviser'
```

La fonction “Eq” (Egal) is where the “magic” happens. Il sera facile pour vous de comprendre le code suivant maintenant.

```
            elif fonction == 'Egal':
                if self.FonctionCourante == 'Ajouter':
                    self.ValeurCourante += self.ValeurAncienne
                elif self.FonctionCourante == 'Soustraire':
                    self.ValeurCourante = self.ValeurAncienne - self.ValeurCourante
                elif self.FonctionCourante == 'Multiplier':
                    self.ValeurCourante *= self.ValeurAncienne
                elif self.FonctionCourante == 'Diviser':
                    self.ValeurCourante = self.ValeurAncienne / self.ValeurCourante
                self.Rafraichir()
                self.ValeurCourante = 0
                self.ValeurAncienne = 0
```

utilisant cet opérateur magique, nous obtenons une simple fonction « élévation à la puissance ». Par exemple 10<nowiki>**2 renvoie 100. 10<nowiki></ nowiki>-2 retourne 0.01. Parfois, en utilisant cette routine, cela conduit à un problème d'arrondi, mais pour notre calculatrice simple, cela fonctionnera pour la plupart des nombres décimaux raisonnables. Je vais vous laisser le soin de travailler à une meilleure fonction. Prenez cela comme vos devoirs pour ce mois-ci.

```
def funcClear(self):

self.CurrentValue = 0

self.HolderValue = 0

self.DisplayIt()
```

La routine « foncEffacer » efface simplement les deux variables mémoire, puis rafraîchit l'affichage. def foncEffacer(self): self.ValeurCourante = 0 self.ValeurAncienne = 0 self.Rafraichir() Maintenant les fonctions. Nous avons déjà discuté de ce qui se passe avec la fonction « Dec ». Nous l'avons traitée en premier avec l'instruction « if ». Nous allons passer au « else » et, dans le cas où la fonction est autre, nous effaçons les variables « self.PartieDecimale » et « self.CompteDecimales ».

Les prochaines étapes sont indiquées sur la page précédente (encadré de droite).

La routine « Rafraichir » règle simplement la valeur de l'étiquette d'affichage. N'oubliez pas que nous avons dit à l'étiquette de « surveiller » la variable « self.AffichageCourant ». À chaque fois que cette variable change, l'étiquette change automatiquement d'affichage pour correspondre. Nous utilisons la méthode « .set » pour changer la valeur.

```
def Rafraichir(self):

print('ValeurCourante = {0}
- ValeurAncienne =
{1}'.format(self.ValeurCoura
nte,self.ValeurAncienne))

self.AffichageCourant.set(se
lf.ValeurCourante)
```

Enfin, nous avons nos lignes de démarrage.

```
if __name__ == '__main__':

Demarrage()
```

Maintenant, vous pouvez exécuter le programme et l'essayer.

Comme toujours, le code de cet article peut être trouvé sur PasteBin. Les exemples 1, 2 et 3 sont ici : http://pastebin.com/RAF4KK6E et l'exemple Calc.py est ici : http://pastebin.com/Pxr0H8FJ

Le mois prochain, nous allons continuer à explorer Tkinter et la richesse de ses widgets. Dans un prochain article, nous verrons un concepteur d'interface graphique pour Tkinter appelé PAGE. En attendant, amusez-vous bien. Je pense que vous apprécierez Tkinter.

TUTORIEL

Programmer en Python - Partie 26

Greg Walters

Le mois dernier, nous avons parlé de TkInter et de quatre des widgets disponibles : la fenêtre principale, les fenêtres, les boutons et les étiquettes (ou labels). Je vous ai également dit le mois dernier que je parlerais de la façon d'avoir un widget autre que le widget de premier niveau comme parent. Aussi, ce mois-ci, nous allons approfondir les fenêtres, les boutons et les étiquettes, et introduire les cases à cocher, les boutons radio, les zones de texte (ou widgets Entry), les listes avec une barre de défilement verticale (ListBox) et les fenêtres de message. Avant de commencer, examinons certains de ces widgets.

Les cases à cocher servent à faire plusieurs choix parmi plusieurs propositions et ont deux états : cochée ou non cochée, ou on pourrait dire aussi oui ou non. Elles sont généralement utilisées pour fournir une série d'options où une, quelques-unes ou toutes peuvent être sélectionnées. Vous pouvez définir un événement pour vous informer quand la case a changé d'état ou tout simplement pour interroger la valeur du widget à tout moment.

Les boutons radio servent à faire un choix parmi plusieurs propositions. Ils ont aussi deux états, oui ou non. Cependant, ils sont groupés ensemble pour fournir un groupe d'options dont une seule peut être choisie. Vous pouvez avoir plusieurs groupes de boutons radio qui, s'ils sont bien programmés, n'interféreront pas entre eux.

Une ListBox fournit une liste d'éléments parmi lesquels l'utilisateur peut choisir. La plupart du temps, vous voulez que l'utilisateur sélectionne un seul des éléments à la fois, mais, parfois, vous pouvez vouloir permettre à l'utilisateur de sélectionner plusieurs éléments. Une barre de défilement peut être placée horizontalement ou verticalement afin de permettre à l'utilisateur de parcourir facilement tous les éléments disponibles.

Notre projet consistera en une fenêtre principale et sept cadres principaux qui regrouperont visuellement nos ensembles de widgets :

1. Le premier cadre sera très basique : il contient simplement différents labels, montrant les différentes options de relief.
2. Le second contiendra des boutons - c'est plutôt simple aussi - qui utilisent différentes options de relief.
3. Dans ce cadre, nous aurons deux cases à cocher et un bouton qui peut les activer/désactiver et qui enverront leur état (1 ou 0) à la fenêtre du terminal lorsqu'on clique dessus ou les active/désactive.
4. Ensuite, nous aurons deux groupes de trois boutons radio, chacun envoyant un message à la fenêtre du terminal lorsqu'on clique dessus. Chaque groupe est indépendant de l'autre.
5. Celui-ci contient des champs de texte qui ne sont pas nouveaux pour vous, mais il y a aussi un bouton pour activer et désactiver l'un d'eux. Lorsqu'il est désactivé, aucune saisie ne peut y être faite.
6. Celui-ci contient une liste avec une barre de défilement verticale qui envoie un message au terminal chaque fois qu'un élément est sélectionné ; il aura deux boutons. Un bouton va effacer la zone de liste et l'autre la remplira avec des valeurs fictives.
7. Le dernier cadre contient une série de boutons qui appellent les différents types de boîtes de message.

```
# widgetdemo1.py
# Labels
from Tkinter import *

class Demo:
  def __init__(self,principale):
      self.DefinirVariables()
       f = self.ConstruireWidgets(principale)
       self.PlacerWidgets(f)
```

Bon, maintenant nous allons commencer notre projet. Nommons-le « wid-getdemo1.py ». Assurez-vous de le sauvegarder, car nous allons écrire notre projet par petits morceaux et construire notre application complète petit à petit. Chaque morceau tourne autour de l'un des cadres. Vous remarquerez que j'intègre un certain nombre de commentaires au fur et à mesure, pour que vous puissiez suivre ce qui se passe. Voici les premières lignes (voir encadré ci-dessus).

```
def ConstruireWidgets(self,principale):
      # definition de nos widgets
      fenetre = Frame(principale)
      # labels (ou etiquettes)
      self.fenetreLabels = Frame(fenetre,relief = SUNKEN,padx = 3, pady = 3,
                                 borderwidth = 2, width = 500)
      self.lbl1 = Label(self.fenetreLabels,text="Label plat",relief = FLAT,
                                 width = 13,borderwidth = 2)
      self.lbl2 = Label(self.fenetreLabels,text="Label creux", relief = SUNKEN,
                                 width = 13, borderwidth = 2)
      self.lbl3 = Label(self.fenetreLabels,text="Label arete", relief = RIDGE, width = 13,
                        borderwidth = 2)
      self.lbl4 = Label(self.fenetreLabels,text="Label souleve", relief = RAISED,
                        width = 13, borderwidth = 2)
      self.lbl5 = Label(self.fenetreLabels,text="Label rainure", relief = GROOVE,
                        width = 13, borderwidth = 2)
      return fenetre
```

Les deux premières lignes (commentaires) sont le nom de l'application et le thème de cette partie. La ligne trois est notre déclaration d'importation. Ensuite, nous définissons notre classe. La ligne suivante commence notre routine __init__, avec laquelle vous devriez tous être familiers maintenant ; mais si vous venez juste de nous rejoindre, c'est le code qui est exécuté quand on instancie la routine dans la partie principale du programme. Nous lui passons la fenêtre racine (ou toplevel), qui s'appelle « principale » ici. Les trois dernières lignes (jusqu'à présent) appellent trois routines différentes. La première (DefinirVariables) réglera différentes variables dont nous aurons besoin plus tard. La suivante (ConstruireWidgets) sera l'endroit où nous définissons nos widgets, et la dernière (PlacerWidgets) est celle où nous allons placer les widgets dans la fenêtre racine. Comme nous l'avons fait la dernière fois, nous allons utiliser le gestionnaire de géométrie « grille ». Notez que ConstruireWidgets retournera l'objet « f » (qui est notre fenêtre racine) et que nous le passerons à la routine PlacerWidgets.

Voici notre routine ConstruireWidgets (ci-contre, en haut à droite). Les lignes qui commencent par « self. » ont été coupées pour deux raisons. Tout d'abord, c'est une bonne pratique de garder la longueur de la ligne à moins de 80 caractères. Deuxièmement, cela facilite les choses pour notre merveilleux éditeur. Vous avez deux possibilités : soit écrire des lignes longues, soit les garder comme ça. Python nous permet de couper les lignes tant qu'elles sont dans des parenthèses ou des crochets. Comme je l'ai dit précédemment, nous définissons les widgets avant de les placer dans la grille. Quand nous écrirons la routine suivante, vous remarquerez que nous pouvons aussi définir un widget au moment où nous le plaçons dans la grille, mais le définir avant de le mettre dans la grille dans une routine comme celle-ci facilite les choses, puisque nous faisons (la plupart) des définitions dans cette routine.

Nous définissons donc d'abord notre fenêtre principale. C'est là que nous mettrons le reste de nos widgets. Ensuite, nous définissons une fenêtre fille (de la fenêtre principale), qui contiendra cinq étiquettes, et l'appelons fenetreLabels. Nous réglons les différents attributs de la fenêtre ici. Nous réglons le relief à « en creux » (« SUNKEN »), un remplissage de 3 pixels à gauche et à droite (padx), et de 3 pixels en haut et en bas (pady). Nous avons également mis la largeur de bordure à 2 pixels de telle sorte que son relief en creux soit perceptible. Par défaut, la largeur de bordure vaut 0 et l'effet de creux ne serait pas visible. Enfin, nous avons mis la largeur totale de la fenêtre à 500 pixels.

Ensuite, nous définissons chaque widget étiquette que nous allons utiliser. Nous fixons le parent à self.fenetreLabels, et non pas fenetre. De cette façon, toutes les étiquettes sont des enfants de fenetreLabels et fenetreLabels est un enfant de fenetre. Remarquez que chaque définition est à peu près semblable pour l'ensemble des cinq étiquettes, sauf le nom du widget (lbl1, lbl2, etc), le texte et le relief ou l'effet visuel. Enfin, nous retour-

nons la fenêtre à la routine appelante (_init_).

Voici notre routine PlacerWidgets (page suivante, en haut à droite).

Nous récupérons l'objet fenêtre en tant que paramètre appelé « principale ». Nous l'assignons à « fenetre » simplement pour être cohérent avec ce que nous avons fait dans la routine ConstruireWidgets. Ensuite, nous mettons en place la grille principale (fenetre.grid (column=0, row=0)). Si nous ne faisons pas cela, rien ne fonctionnera correctement. Ensuite, nous commençons à mettre nos widgets dans les emplacements de la grille. D'abord nous mettons la fenêtre (fenetreLabels) qui contient toutes nos étiquettes et définissons ses attributs. Nous la plaçons colonne 0, ligne 1, réglons le remplissage à 5 pixels sur tous les côtés, lui disons de s'étaler sur 5 colonnes (à droite et à gauche), et enfin utilisons l'attribut « sticky » [Ndt : collant] pour forcer la fenêtre à s'étendre complètement à gauche et à droite (« WE » pour Ouest et Est). Maintenant vient la partie qui enfreint la règle dont je vous ai parlé. Nous mettons une étiquette comme premier widget dans la fenêtre, mais nous ne l'avons pas défini à l'avance : nous le définissons maintenant. Nous avons mis comme parent fenetreLabels, tout comme les autres étiquettes. Nous réglons le texte à « 'La-bels |' », la largeur à 15, et l'ancre à Est ('e'). Si vous vous souvenez de la dernière fois, en utilisant l'attribut d'ancrage, nous pouvons choisir où le texte s'affiche dans le widget. Dans ce cas, c'est le long du bord droit. Maintenant la partie amusante. Ici, nous définissons l'emplacement de la grille (et tous les autres attributs de la grille dont nous avons besoin), simplement en ajoutant « .grid » à la fin de la définition des étiquettes.

```
def PlacerWidgets(self, principale):
    fenetre = principale
    # place les widgets
    fenetre.grid(column = 0, row = 0)
    # place les labels
    self.fenetreLabels.grid(column = 0, row = 1, padx = 5, pady = 5,
                            columnspan = 5,sticky='WE')
    l = Label(self.fenetreLabels,text='Labels |',width=15,
              anchor='e').grid(column=0,row=0)
    self.lbl1.grid(column = 1, row = 0, padx = 3, pady = 5)
    self.lbl2.grid(column = 2, row = 0, padx = 3, pady = 5)
    self.lbl3.grid(column = 3, row = 0, padx = 3, pady = 5)
    self.lbl4.grid(column = 4, row = 0, padx = 3, pady = 5)
    self.lbl5.grid(column = 5, row = 0, padx = 3, pady = 5)
```

Ensuite, nous plaçons toutes nos autres étiquettes dans la grille, à partir de la colonne 1, ligne 0.

Voici notre routine DefinirVariables. Notez que nous utilisons simplement l'instruction pass pour l'instant. Nous la remplirons plus tard, car nous n'en avons pas besoin pour cette partie :

```
def DefinirVariables(self):
  # Definit nos ressources
  pass
```

Et enfin nous plaçons notre code pour la routine principale :

```
root = Tk()
root.geometry('750x40+150+150')
root.title("Widget Demo 1")
demo = Demo(root)
root.mainloop()
```

D'abord, on initialise une instance de Tk. Puis nous définissons la taille de la fenêtre principale à 750 pixels de large sur 40 pixels de haut et la localisons à 150 pixels de la gauche et du haut de l'écran. Puis nous réglons le titre de la fenêtre et instancions notre objet Demo et, enfin, appelons la boucle principale de Tk.

Essayez. Vous devriez voir les cinq étiquettes ainsi que l'étiquette de « dernière minute » avec divers effets magnifiques.

Les boutons

Maintenant, enregistrez ce que vous avez en tant que widgetde-mo1a.py

```
# place les boutons
self.fenetreBoutons.grid(column=0, row = 2, padx = 5,
                         pady = 5, columnspan = 5,sticky = 'WE')
l = Label(self.fenetreBoutons,text='Boutons |',width=15,
          anchor='e').grid(column=0,row=0)
self.btn1.grid(column = 1, row = 0, padx = 3, pady = 3)
self.btn2.grid(column = 2, row = 0, padx = 3, pady = 3)
self.btn3.grid(column = 3, row = 0, padx = 3, pady = 3)
self.btn4.grid(column = 4, row = 0, padx = 3, pady = 3)
self.btn5.grid(column = 5, row = 0, padx = 3, pady = 3)
```

et nous allons ajouter quelques boutons. Puisque nous avons construit notre programme de base ainsi, nous allons simplement pouvoir y ajouter les parties qui manquent. Commençons par la routine ConstruireWidgets. Après les définitions des étiquettes, et avant le « return fenetre », ajoutez ce qui se trouve en haut de la page suivante.

Rien de bien nouveau ici. Nous avons défini les boutons avec leurs attributs et avons fixé leurs fonctions de rappel avec un « .bind ». Notez que nous utilisons lambda pour envoyer les valeurs 1 à 5 suivant le bouton sur lequel on clique. Dans la fonction de rappel, nous allons utiliser cela afin de savoir quel bouton on doit gérer. Maintenant, nous allons travailler dans la routine PlacerWidgets. Placez le code (page précédente, en bas à droite) juste après l'emplacement de la dernière étiquette.

Une fois de plus, rien de vraiment nouveau ici, donc nous allons continuer. Voici notre routine de rappel (ci-contre, en bas à droite). Placez-la après la routine DefinirVariables.

Encore une fois, rien de vraiment sensationnel ici. Nous utilisons simplement une série de routines IF/ELIF pour afficher quel bouton a été cliqué. La principale chose à regarder ici (lorsque nous exécutons le programme) est que le bouton « en creux » ne bouge pas lorsqu'on clique dessus. En général on n'utilise pas le relief « en creux », sauf si vous souhaitez un bouton qui reste enfoncé lorsque vous cliquez dessus. Enfin, nous avons besoin d'ajuster la déclaration de la géométrie à cause des widgets supplémentaires que nous avons ajoutés :

```
root.geometry('750x110+150+150')
```

Ok. C'est terminé pour celui-ci. Enregistrez-le et lancez-le.

Maintenant sauvegardez ceci comme widgetdemo1b.py et nous allons passer aux cases à cocher.

Les cases à cocher

Comme je l'ai dit précédemment, cette partie de la démo a un bouton normal et deux cases à cocher. L'apparence de la première case est celle, normale, à laquelle vous pouvez vous attendre. La seconde est plus comme un « bouton collant » - quand elle n'est pas sélectionnée (ou cochée), elle ressemble à un bouton normal. Lorsque vous la sélectionnez, elle ressemble à un bouton qui reste enfoncé. Nous pouvons faire cela simplement en définissant l'attribut indicatoron à False. Le bouton « normal » permet de basculer les cases de « cochées » à « décochées » et vice- versa, à chaque fois que l'on clique dessus. Nous arrivons à pro-

```
# boutons
 self.fenetreBoutons = Frame(fenetre,relief = SUNKEN,padx = 3,
                             pady = 3, borderwidth = 2, width = 500)
 self.btn1 = Button(self.fenetreBoutons,text="Bouton plat",
                    relief = FLAT, borderwidth = 2)
 self.btn2 = Button(self.fenetreBoutons,text="Bouton creux",
                    relief = SUNKEN, borderwidth = 2)
 self.btn3 = Button(self.fenetreBoutons,text="Bouton arete",
                    relief = RIDGE, borderwidth = 2)
 self.btn4 = Button(self.fenetreBoutons,text="Bouton souleve",
                    relief = RAISED, borderwidth = 2)
 self.btn5 = Button(self.fenetreBoutons,text="Bouton rainure",
                    relief = GROOVE, borderwidth = 2)
 self.btn1.bind('<ButtonRelease-1>',lambda e: self.clicBouton(1))
 self.btn2.bind('<ButtonRelease-1>',lambda e: self.clicBouton(2))
 self.btn3.bind('<ButtonRelease-1>',lambda e: self.clicBouton(3))
 self.btn4.bind('<ButtonRelease-1>',lambda e: self.clicBouton(4))
 self.btn5.bind('<ButtonRelease-1>',lambda e: self.clicBouton(5))
```

```
def clicBouton(self,val):
    if val == 1:
        print("Clic bouton plat...")
    elif val == 2:
        print("Clic bouton creux...")
    elif val == 3:
        print("Clic bouton arete...")
    elif val == 4:
        print("Clic bouton souleve...")
    elif val == 5:
        print("Clic bouton rainure...")
```

grammer cela en appelant la méthode .toggle liée à la case à cocher. Nous relions l'événement clic gauche de la souris (lorsque le bouton est relâché) à une fonction afin de pouvoir envoyer un message (dans notre cas) au terminal. En plus de tout cela, nous mettons en place deux variables (une pour chacune des cases à cocher) que l'on peut interroger à tout moment. Ici, nous interrogeons ces valeurs et les affichons à chaque fois qu'une case est cliquée. Faites attention à la partie variable du code : elle est utilisée dans de nombreux widgets. Dans la routine ConstruireWidgets, après le code des boutons que nous venons d'ajouter et avant l'instruction de retour, placez le code (ci-contre, en haut à droite).

Encore une fois, vous avez vu tout cela avant. Nous créons la fenêtre pour contenir nos widgets. Nous créons un bouton et deux cases à cocher. Plaçons-les maintenant (ci-contre, au milieu à droite).

Maintenant, nous définissons les deux variables que nous allons utiliser pour surveiller la valeur de chaque case à cocher. Sous DefinirVariables, commentez l'instruction pass et ajoutez ceci :

```
self.Chk1Val =
IntVar()
self.Chk2Val =
IntVar()
```

Après la fonction de rappel des boutons, placez ce qui suit (ci-contre, en bas à droite).

Et enfin remplacez l'instruction de géométrie par ceci :

```
root.geometry('75
0x170+150+150')
```

Enregistrez et exécutez. Enregistrez-le comme widgetdemo1c.py et continuons avec les boutons radio.

```
# checkbox (ou cases a cocher)
self.fenetreCases = Frame(fenetre, relief = SUNKEN, padx = 3, pady = 3,
                          borderwidth = 2, width = 500)
self.chk1 = Checkbutton(self.fenetreCases, text = "Case a cocher normale",
                        variable=self.Chk1Val)
self.chk2 = Checkbutton(self.fenetreCases, text = "Case a cocher",
                        variable=self.Chk2Val,indicatoron = False)
self.chk1.bind('<ButtonRelease-1>',lambda e: self.clicCases(1))
self.chk2.bind('<ButtonRelease-1>',lambda e: self.clicCases(2))
self.btnInverserCases = Button(self.fenetreCases,text="Inverser cases")
self.btnInverserCases.bind('<ButtonRelease-1>',self.btnInverser)
```

```
# place les cases à cocher et le bouton d'inversion
self.fenetreCases.grid(column = 0, row = 3, padx = 5, pady = 5,
                       columnspan = 5,sticky = 'WE')
l = Label(self.fenetreCases,text='Cases à cocher |',width=15,
          anchor='e').grid(column=0,row=0)
self.btnInverserCases.grid(column = 1, row = 0, padx = 3, pady = 3)

self.chk1.grid(column = 2, row = 0, padx = 3, pady = 3)
self.chk2.grid(column = 3, row = 0, padx = 3, pady = 3)
```

```
def btnInverser(self,p1):
    self.chk1.toggle()
    self.chk2.toggle()
    print("Valeur de la case à cocher 1 : {0}".format(self.Chk1Val.get()))
    print("Valeur de la case à cocher 2 : {0}".format(self.Chk2Val.get()))
```

Les boutons radio

Si vous êtes assez vieux pour vous souvenir des autoradios avec boutons poussoirs pour sélectionner les stations pré-réglées, vous comprendrez pourquoi on appelle cela des boutons radio. Lorsque vous utilisez des boutons radio, l'attribut variable est très important. C'est ce qui regroupe les boutons radio ensemble. Dans cette démo, le premier groupe de boutons est formé avec la variable nommée self.RBVal. Le second groupe est formé par la variable self.RBVal2. Nous devons également définir l'attribut « value » au moment de la conception, afin de garantir que les boutons retourneront une valeur qui a du sens quand ils seront cliqués.

Retournez dans ConstruireWidgets, et ajoutez le code (ci-dessous), juste avant l'instruction de retour.

```
        # boutons radio
        self.fenetreBoutonsRadio = Frame(fenetre, relief = SUNKEN, padx = 3, pady = 3, borderwidth = 2, width = 500)
        self.rb1 = Radiobutton(self.fenetreBoutonsRadio, text = "Radio 1", variable = self.RBVal, value = 1)
        self.rb2 = Radiobutton(self.fenetreBoutonsRadio, text = "Radio 2", variable = self.RBVal, value = 2)
        self.rb3 = Radiobutton(self.fenetreBoutonsRadio, text = "Radio 3", variable = self.RBVal, value = 3)
        self.rb1.bind('<ButtonRelease-1>',lambda e: self.clicBoutonRadio())
        self.rb2.bind('<ButtonRelease-1>',lambda e: self.clicBoutonRadio())
        self.rb3.bind('<ButtonRelease-1>',lambda e: self.clicBoutonRadio())
        self.rb4 = Radiobutton(self.fenetreBoutonsRadio, text = "Radio 4", variable = self.RBVal2, value = "1-1")
        self.rb5 = Radiobutton(self.fenetreBoutonsRadio, text = "Radio 5", variable = self.RBVal2, value = "1-2")
        self.rb6 = Radiobutton(self.fenetreBoutonsRadio, text = "Radio 6", variable = self.RBVal2, value = "1-3")
        self.rb4.bind('<ButtonRelease-1>',lambda e: self.clicBoutonRadio2())
        self.rb5.bind('<ButtonRelease-1>',lambda e: self.clicBoutonRadio2())
        self.rb6.bind('<ButtonRelease-1>',lambda e: self.clicBoutonRadio2())
```

Ajoutez ceci dans PlacerWidgets :

```
        # place les boutons radio et selectionne le premier
        self.fenetreBoutonsRadio.grid(column = 0, row = 4, padx = 5, pady = 5, columnspan = 5,sticky = 'WE')
        l = Label(self.fenetreBoutonsRadio,
                  text='Boutons radio |',
                  width=15,anchor='e').grid(column=0,row=0)
        self.rb1.grid(column = 2, row = 0, padx = 3, pady = 3, sticky = 'EW')
        self.rb2.grid(column = 3, row = 0, padx = 3, pady = 3, sticky = 'WE')
        self.rb3.grid(column = 4, row = 0, padx = 3, pady = 3, sticky = 'WE')
        self.RBVal.set("1")
        l = Label(self.fenetreBoutonsRadio,text='| Un autre groupe |',
                  width = 15,
                  anchor = 'e').grid(column = 5, row = 0)
        self.rb4.grid(column = 6, row = 0)
        self.rb5.grid(column = 7, row = 0)
        self.rb6.grid(column = 8, row = 0)
        self.RBVal2.set("1-1")
```

Une chose à noter ici. Remarquez les définitions de « dernière minute » pour les étiquettes dans la routine PlacerWidgets. Ces lignes longues sont coupées pour montrer comment utiliser les parenthèses pour permettre à nos longues lignes d'être formatées correctement dans notre code, et de fonctionner toujours correctement.

Dans DefinirVariables, ajoutez :

```
self.RBVal = IntVar()
```

Ajoutez les routines de clics :

```
def clicBoutonRadio(self):
      print("Clic bouton
radio - Valeur :
{0}".format(self.RBVal.get()))

  def clicBoutonRadio2
(self):
      print("Clic bouton
radio - Valeur :
{0}".format(self.RBVal2.get()))
```

et enfin modifiez à nouveau la géométrie comme ceci :

```
root.geometry('750x220+150+150')
```

Enregistrez le projet sous widgetdemo1d.py et exécutez-le. Maintenant, nous allons travailler sur les champs de texte standard (ou widgets de saisie).

Les champs de texte

Encore une fois, nous avons déjà utilisé des champs de texte (ou widgets de saisie) dans diverses interfaces graphiques auparavant. Mais cette fois-ci, comme je l'ai dit précédemment, nous allons montrer comment empêcher l'utilisateur de faire des changements dans le champ de texte en le désactivant. Cela s'avère utile si vous affichez certaines données et permettez à l'utilisateur de les modifier seulement quand il est dans un mode d'édition. Maintenant vous devriez savoir que la première chose que nous devons faire est d'ajouter du code à la routine ConstruireWidgets (ci-contre, à droite).

Comme d'habitude, nous créons notre fenêtre. Puis nous créons notre

```
      # champs de texte
      self.fenetreChampsTexte = Frame(fenetre, relief = SUNKEN, padx
= 3, pady = 3, borderwidth = 2, width = 500)
      self.txt1 = Entry(self.fenetreChampsTexte, width = 10)
      self.txt2 = Entry(self.fenetreChampsTexte,
disabledbackground="#cccccc", width = 10)
      self.btnDesactiver = Button(self.fenetreChampsTexte, text =
"Activer/Desactiver")
      self.btnDesactiver.bind('<ButtonRelease-1>',
self.clicBoutonDesactiver)
```

Ensuite, ajoutez ces lignes de code à la routine PlacerWidgets :

```
     # place les champs de texte
      self.fenetreChampsTexte.grid(column = 0, row = 5, padx = 5,
pady = 5, columnspan = 5,sticky = 'WE')
      l = Label(self.fenetreChampsTexte,text='Champs de texte
|',width=15, anchor='e').grid(column=0,row=0)
      self.txt1.grid(column = 2, row = 0, padx = 3, pady = 3)
      self.txt2.grid(column = 3, row = 0, padx = 3, pady = 3)
      self.btnDesactiver.grid(column = 1, row = 0, padx = 3, pady = 3)
```

Ajoutez cette ligne en bas de la routine DefinirVariables :

```
       self.Disabled = False
```

Maintenant ajoutez la fonction qui répond au clic sur le bouton :

```
  def clicBoutonDesactiver(self,p1):
      if self.Disabled == False:
          self.Disabled = True
          self.txt2.configure(state='disabled')
      else:
          self.Disabled = False
          self.txt2.configure(state='normal')
```

Enfin, relancez la ligne de géométrie ::

```
root.geometry('750x270+150+150')
```

Sauvegardez-le sous le nom widgetdemo1d.py et exécutez-le.

barre de défilement verticale. Nous faisons cela avant de créer la liste, parce que nous devons faire référence à la méthode « .set » de la barre de défilement. Remarquez l'attribut « height = 5 ». Cela force la liste à montrer 5 éléments à la fois. Dans la déclaration .bind, nous utilisons « "ListboxSelect" » comme événement. C'est ce qu'on appelle un événement virtuel, puisque ce n'est pas vraiment un événement « officiel ».

Maintenant, nous allons nous occuper du code supplémentaire dans la routine PlacerWidgets (page suivante, encadré de gauche).

Les boîtes de dialogue

Cette section est tout simplement une série de boutons « normaux » qui appellent les différents types de boîtes de dialogue. Nous les avons déjà rencontrés avec une boîte à outils différente. Nous allons explorer seulement 5 types différents, mais il y en a plus. Dans cette section, nous allons regarder Information, Avertissement, Erreur, Question, et les dialogues Oui/Non. Ils sont très utiles lorsque vous avez besoin de faire passer des informations à votre utilisateur d'une manière assez importante. Dans la routine ConstruireWidgets, ajoutez (ci-contre, en bas à droite).

Voici la routine d'appui (ci-contre, en haut à droite). Pour les trois premiers (Info, Avertissement et Erreur), il suffit d'appeler « tkMessageBox.show-info », ou celui dont vous avez besoin, avec deux paramètres. Le premier est le titre de la boîte de message et le second est le message réel que vous voulez montrer. L'icône est gérée pour vous par Tkinter. Pour les dialogues qui fournissent une réponse (Question, Oui/Non), nous fournissons une variable qui reçoit la valeur correspondant au bouton cliqué. Dans le cas de la boîte de dialogue Question, la réponse est « Oui » ou « Non », et dans le cas du dialogue Oui/Non, la réponse est « True »

```
      # champs de texte
      self.fenetreChampsTexte = Frame(fenetre, relief = SUNKEN, padx
= 3, pady = 3, borderwidth = 2, width = 500)
      self.txt1 = Entry(self.fenetreChampsTexte, width = 10)
      self.txt2 = Entry(self.fenetreChampsTexte,
disabledbackground="#cccccc", width = 10)
      self.btnDesactiver = Button(self.fenetreChampsTexte, text =
"Activer/Desactiver")
      self.btnDesactiver.bind('<ButtonRelease-1>',
self.clicBoutonDesactiver)
```

Ensuite, ajoutez ces lignes de code à la routine PlacerWidgets :

```
     # place les champs de texte
      self.fenetreChampsTexte.grid(column = 0, row = 5, padx = 5,
pady = 5, columnspan = 5,sticky = 'WE')
      l = Label(self.fenetreChampsTexte,text='Champs de texte
|',width=15, anchor='e').grid(column=0,row=0)
      self.txt1.grid(column = 2, row = 0, padx = 3, pady = 3)
      self.txt2.grid(column = 3, row = 0, padx = 3, pady = 3)
      self.btnDesactiver.grid(column = 1, row = 0, padx = 3, pady = 3)
```

Ajoutez cette ligne en bas de la routine DefinirVariables :

```
       self.Disabled = False
```

Maintenant ajoutez la fonction qui répond au clic sur le bouton :

```
  def clicBoutonDesactiver(self,p1):
      if self.Disabled == False:
          self.Disabled = True
          self.txt2.configure(state='disabled')
      else:
          self.Disabled = False
          self.txt2.configure(state='normal')
```

Enfin, relancez la ligne de géométrie ::

```
root.geometry('750x270+150+150')
```

Sauvegardez-le sous le nom widgetdemo1d.py et exécutez-le.

```
       # place la liste et les boutons associes
      self.fenetreListe.grid(column = 0, row = 6, padx = 5,
pady = 5, columnspan = 5,sticky = 'WE')
      l = Label(self.fenetreListe,text='Liste |',width=15,
anchor='e').grid(column=0,row=0,rowspan=2)
      self.liste.grid(column = 2, row = 0,rowspan=2)
      self.defilementV.grid(column = 3, row = 0,rowspan =
2, sticky = 'NSW')
      self.btnEffacerListe.grid(column = 1, row = 0, padx = 5)
      self.btnRemplirListe.grid(column = 1, row = 1, padx = 5)
```

Ajoutez ceci dans DefinirVariables :

```
     # les elements pour notre liste
      self.exemples = ['Element un','Element deux','Element
trois','Element quatre']
```

Et ajoutez les routines de support suivantes :

```
  def effacerListe(self):
      self.liste.delete(0,END)

  def remplirListe(self):
      # Note : effacer d'abord la liste ; aucune
verification n'est faite
      for ex in self.exemples:
          self.liste.insert(END,ex)
      # insert([0,ACTIVE,END],element)

  def listeSelection(self,p1):
      print("Clic sur un élément de la liste")
      items = self.liste.curselection()
      selitem = items[0]
      print("Index de l'élément choisi :
{0}".format(selitem))
      print("Texte de l'élément choisi :
{0}".format(self.liste.get(selitem)))
```

Enfin, mettez à jour la ligne de géométrie :

```
root.geometry('750x370+150+150')
```

Sauvegarder cela comme widgetdemo1e.py et exécutez-le. Maintenant, nous allons faire les dernières modifications à notre application.

```
      # des choses pour la liste
      self.fenetreListe = Frame(fenetre,
                                relief = SUNKEN,
                                padx = 3,
                                pady = 3,
                                borderwidth = 2,
                                width = 500
      )
      # boite avec barre de défilement pour la
liste
      self.defilementV =
Scrollbar(self.fenetreListe)
      self.liste = Listbox(self.fenetreListe,
                           height = 5,
                           yscrollcommand =
self.defilementV.set)
# hauteur par defaut = 10

self.liste.bind('<<ListboxSelect>>',self.listeSelec
tion)
      self.defilementV.config(command =
self.liste.yview)
      self.btnEffacerListe = Button(
           self.fenetreListe,
           text = "Effacer liste",
           command = self.effacerListe,
           width = 11
      )
      self.btnRemplirListe = Button(
           self.fenetreListe,
           text = "Remplir liste",
           command = self.remplirListe,
           width = 11
      )
      # <<ListboxSelect>> est un evenement virtuel
      # remplit la liste
      self.remplirListe()
```

ou « False ».

Enfin, modifiez la ligne de géométrie :

```
root.geometry('750x490+550+150')
```

Sauvegardez ceci sous le nom widgetdemo1f.py et amusez-vous avec.

J'ai placé le code de widgetdemo1f.py sur pastebin ici :
http://pastebin.com/TQgppVnF

C'est tout pour cette fois. J'espère que ceci vous aura inspiré à explorer toutes les fonctionnalités proposées par tkinter. À la prochaine fois.

```
  def afficheFenetreMessage(self,which):
      if which == 1:
          tkMessageBox.showinfo('Demo','Voici un message INFO')
      elif which == 2:
          tkMessageBox.showwarning('Demo','Voici un message WARNING (avertissement)')
      elif which == 3:
          tkMessageBox.showerror('Demo','Voici un message ERREUR')
      elif which == 4:
          rep = tkMessageBox.askquestion('Demo','Voici une QUESTION ?')
          print('clic sur {0}...'.format(rep))
      elif which == 5:
          rep = tkMessageBox.askyesno('Demo','Voici un message OUI/NON')
          print('clic sur {0}...'.format(rep))
```

```
# boutons pour afficher les fenetres de messages et de dialogues
self.fenetreMessages = Frame(fenetre,relief = SUNKEN,padx = 3, pady = 3, borderwidth = 2)
self.btnMBInfo = Button(self.fenetreMessages,text = "Info")
self.btnMBWarning = Button(self.fenetreMessages,text = "Avertissement")
self.btnMBError = Button(self.fenetreMessages,text = "Erreur")
self.btnMBQuestion = Button(self.fenetreMessages,text = "Question")
self.btnMBYesNo = Button(self.fenetreMessages,text = "Oui/Non")
self.btnMBInfo.bind('<ButtonRelease-1>', lambda e: self.afficheFenetreMessage(1))
self.btnMBWarning.bind('<ButtonRelease-1>', lambda e: self.afficheFenetreMessage(2))
self.btnMBError.bind('<ButtonRelease-1>', lambda e: self.afficheFenetreMessage(3))
self.btnMBQuestion.bind('<ButtonRelease-1>', lambda e: self.afficheFenetreMessage(4))
self.btnMBYesNo.bind('<ButtonRelease-1>', lambda e: self.afficheFenetreMessage(5))
```

Maintenant ajoutez le code dans la routine PlacerWidgets :

```
# boutons de boîtes de messages et de dialogues
self.fenetreMessages.grid(column = 0,row = 7, columnspan = 5, padx = 5, sticky = 'WE')
l = Label(self.fenetreMessages,text='Messages |',width=15, anchor='e').grid(column=0,row=0)
self.btnMBInfo.grid(column = 1, row = 0, padx= 3)
self.btnMBWarning.grid(column = 2, row = 0, padx= 3)
self.btnMBError.grid(column = 3, row = 0, padx= 3)
self.btnMBQuestion.grid(column = 4, row = 0, padx= 3)
self.btnMBYesNo.grid(column = 5, row = 0, padx= 3)
```

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume cinq

Parties 27 à 31

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

Pour l'instant, il s'agit d'une réédition simple de la série **Programmer en Python**, parties 27 à 31, des numéros 53 à 59 ; et, oui, l'incomparable professeur Python, Greg Walters, a pris quelques jours de congé au cours de cette partie de la série !

Gardez à l'esprit la date de publication ; les versions actuelles du matériel et des logiciels peuvent être différentes de celles illustrées. Il vous est recommandé de bien vérifier la version de votre matériel et des logiciels avant d'essayer d'émuler les tutoriels dans ces numéros spéciaux. Il se peut que vous ayez des logiciels plus récents ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

SiteWeb :
http://www.fullcirclemagazine.org/

Forums :
http://ubuntuforums.org/forumdisplay.php?f=270

IRC : #fullcirclemagazine on chat.freenode.net

Équipe éditoriale :
Rédacteur en chef : Ronnie Tucker (pseudo : RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia (pseudo : admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcast : Robin Catling (pseudo : RobinCatling)
podcast@fullcirclemagazine.org
Dir. comm. : Robert Clipsham (pseudo : mrmonday) -
mrmonday@fullcirclemagazine.org

TUTORIEL

Programmer en Python - Partie 27

Greg Walters

Si vous avez déjà fait la queue pour acheter un billet de cinéma, vous avez été dans une file d'attente. Si vous avez eu à attendre dans les bouchons aux heures de pointe, vous avez été dans une file d'attente. Si vous avez déjà attendu dans un bureau administratif avec l'un de ces petits billets qui dit que vous êtes le numéro 98 et le panneau qui affiche « Numéro actuel : 42 », vous avez été dans une file d'attente.

Dans le monde des ordinateurs, les files d'attente sont très répandues. En tant qu'utilisateur, la plupart du temps vous n'avez pas à vous en préoccuper. Elles sont invisibles pour l'utilisateur. Mais si jamais vous avez à faire face à des événements en temps réel, vous allez finir par avoir à traiter avec elles. Il s'agit simplement de données d'un type ou d'un autre, qui attendent dans la file leur tour d'être traitées. Une fois qu'elles sont dans la file, elles attendent jusqu'à être traitées puis disparaissent. Vous ne pouvez pas connaître la valeur de l'élément de donnée suivant, sauf si vous le sortez de la file d'attente. Vous ne pouvez pas, par exemple, obtenir la valeur du quinzième élément de la file d'attente : il vous faut d'abord accéder aux 14 autres éléments. Une fois qu'un élément est consulté, il sort de la file d'attente. Il a disparu et il n'y a aucun moyen de récupérer les données à moins de les enregistrer dans une variable à long terme.

Il existe plusieurs types de files d'attente. Les plus courantes sont FIFO (« First In, First Out » ou premier entré, premier sorti), LIFO (« Last In, First Out » ou dernier entré, premier sorti), priorité et anneau. Nous parlerons des files d'attente anneau une autre fois.

Les files d'attente FIFO sont celles que nous voyons dans la vie quotidienne. Tous les exemples que j'ai énumérés ci-dessus sont des files d'attente FIFO. La première personne dans la ligne est traitée d'abord, s'en va, puis tout le monde se déplace d'une place dans la ligne. Dans un tampon FIFO, il n'y a pas de limite (sauf celle de la raison) au nombre d'éléments qu'il peut contenir. Ils s'empilent simplement dans l'ordre. Lorsqu'un élément est traité, il est sorti de la file et tous les autres se rapprochent d'une position du début de la file d'attente.

Les files d'attente LIFO sont moins fréquentes dans la vie, mais il existe encore des exemples réels. Celui qui vient tout de suite à l'esprit est l'exemple d'une pile d'assiettes dans votre placard de cuisine. Lorsque les assiettes sont lavées et séchées, elles s'empilent dans le placard. La dernière arrivée sur la pile est la première qui sera réutilisée. Tout le reste attend, peut-être pendant des jours, pour être utilisé. C'est une bonne chose que la file d'attente pour un billet de cinéma soit FIFO, n'est-ce pas ? Comme pour la file d'attente FIFO, en restant dans des tailles raisonnables, il n'y a pas de limite à la taille d'une file d'attente LIFO. Le premier élément entré dans la pile doit attendre que tous les éléments arrivés après lui soient retirés de la mémoire tampon (assiettes retirées de la pile) jusqu'à ce qu'il soit le seul restant.

Les files d'attente prioritaires sont un peu plus difficiles à comprendre du premier coup pour beaucoup de gens. Pensez à une entreprise qui possède une seule imprimante. Tout le monde utilise cette imprimante unique. Les travaux d'impression sont traités par ordre de priorité des départements. La paie a une priorité plus élevée (et heureusement) que, par exemple, vous, un programmeur. Vous avez une priorité plus élevée (et heureusement) que la réceptionniste. En bref, donc, les données qui ont une priorité plus élevée sont traitées et sortent de la file d'attente avant les données qui ont une priorité inférieure.

> Il existe plusieurs types de files d'attente. Les plus courantes sont FIFO (First In, First Out), LIFO (Last In, First Out), Priorité et Anneau.

FIFO

Les files d'attente FIFO sont faciles à visualiser en termes de données. Une liste Python est une représentation mentale facile. Considérez cette liste :

```
[1,2,3,4,5,6,7,8,9,10]
```

Il y a 10 articles dans la liste. En tant que liste, vous y accédez par l'index. Cependant, dans une file d'attente, vous ne pouvez pas accéder aux éléments par leur index. Vous devez traiter avec le prochain dans la file et la liste n'est pas figée. Elle est TRÈS dynamique. Lorsque nous demandons à accéder à l'élément suivant, il est retiré de la file d'attente.

```
import Queue
fifo = Queue.Queue()
for i in range(5):
    fifo.put(i)

while not fifo.empty():
    print fifo.get()
```

Donc, en utilisant l'exemple ci-dessous, vous demandez un élément de la file d'attente. Elle retourne le premier élément (1) et la file d'attente ressemble alors à ceci :

```
[2,3,4,5,6,7,8,9,10]
```

Demandez-en deux de plus et vous obtenez 2, puis 3, et la file d'attente ressemble à ceci :

```
[4,5,6,7,8,9,10]
```

Je suis sûr que vous voyez l'idée. Python fournit une simple bibliothèque, assez étonnamment appelée « Queue » [Ndt : qui signifie file d'attente], qui fonctionne bien pour des files d'attente de petite et moyenne taille, jusqu'à environ 500 éléments. Voici un exemple simple de démonstration (encadré ci-dessus, première colonne).

Dans cet exemple, on initialise la file d'attente (fifo = Queue.Queue()) puis on y place les nombres de 0 à 4 (fifo.put(i)). Nous utilisons ensuite la méthode interne .get() pour retirer des éléments de la file

```
import Queue

fifo = Queue.Queue(12)
for i in range(13):
    if not fifo.full():
        fifo.put(i)

while not fifo.empty():
    print fifo.get()
```

d'attente jusqu'à ce que la file d'attente soit vide, .empty(). Nous obtenons 0,1,2,3,4. Vous pouvez également définir le nombre maximal d'éléments que la file d'attente peut manipuler en l'initialisant avec la taille de la file d'attente comme cela :

```
fifo = Queue.Queue(300)
```

Une fois le nombre maximum d'éléments atteint, la file d'attente bloque toutes les entrées supplémentaires. Cela a cependant pour effet secondaire que le programme semble alors « planté ». La meilleure façon de contourner ce problème est d'utiliser la vérification Queue.full() qui indique si la file est pleine (encadré ci-dessus, deuxième colonne).

Ici, la file d'attente est paramétrée à un maximum de 12 éléments. Lorsque nous ajoutons des éléments dans la file d'attente, nous commençons avec 0 et arrivons à 11. Mais lorsque nous atteignons le nombre 12, le tampon est déjà plein. Puisque nous vérifions si la mémoire tampon est pleine avant d'essayer d'ajouter un élément, le dernier élément est tout simplement rejeté.

Il existe d'autres options, mais elles peuvent causer d'autres effets secondaires, et nous aborderons la question dans un prochain article. Ainsi, la plupart du temps, la voie à suivre est soit d'utiliser une file d'attente sans aucune limite, soit de s'assurer que l'on prévoit plus d'espace dans la file d'attente que ce dont on aura besoin.

LIFO

```
import Queue
lifo = Queue.LifoQueue()
for i in range(5):
    lifo.put(i)
while not lifo.empty():
    print lifo.get()
```

La bibliothèque « Queue » prend également en charge les files d'attente LIFO. Nous allons utiliser la liste ci-dessus comme exemple visuel. Lors de la mise en place de notre file d'attente, elle ressemble à ceci :

```
[1,2,3,4,5,6,7,8,9,10]
```

Si on retire trois éléments de la file d'attente, elle ressemble alors à ceci :

```
[1,2,3,4,5,6,7]
```

N'oubliez pas que dans une file d'attente LIFO, les éléments sont enlevés en commençant par le dernier entré. Voici l'exemple simple modifié pour une file d'attente LIFO (encadré troisième colonne).

Lorsqu'on l'exécute, on obtient 4,3,2,1,0.

Comme pour la file FIFO, vous pouvez régler la taille maximum de la file d'attente et utiliser la vérification .full() pour savoir si elle est pleine.

PRIORITÉ

Même si elle n'est pas souvent utilisée, une file de priorité peut parfois être utile. C'est à peu près la même structure que pour les autres files d'attente, mais nous devons lui passer un tuple qui contient à la fois la priorité et les données. Voici un exemple en utilisant la bibliothèque « Queue » :

```
pq = Queue.PriorityQueue()
pq.put((3,'Moyenne 1'))
pq.put((4,'Moyenne 2'))
pq.put((10,'Basse'))
pq.put((1,'Haute'))

while not pq.empty():
    suiv = pq.get()
    print suiv
    print suiv[1]
```

D'abord on initialise la file d'attente.

Puis nous y plaçons quatre éléments. Remarquez que nous utilisons le format (priorité, données) pour placer nos données. La bibliothèque trie nos données selon un ordre basé sur la valeur de priorité. Quand nous extrayons les données, elles ressortent sous forme de tuple, comme lors de l'insertion. Vous pouvez utiliser l'indice pour accéder aux deux parties du tuple. Voici ce que nous obtenons :

```
(1, 'Haute')
Haute
(3, 'Moyenne 1')
Moyenne 1
(4, 'Moyenne 2')
Moyenne 2
(10, 'Basse')
Basse(
```

Dans nos deux premiers exemples, nous avons simplement affiché les données qui sortent de notre file d'attente. C'est très bien pour ces exemples, mais dans le monde réel de la programmation, vous aurez probablement besoin de faire quelque chose avec cette information dès qu'elle sort de la file d'attente, sinon elle sera perdue. Lorsque nous utilisons « print fifo.get », nous envoyons les données vers le terminal puis elles sont détruites. Il faut juste garder ça à l'esprit.

Maintenant, nous allons utiliser une partie de ce que nous avons déjà appris sur Tkinter pour créer un programme de démo de file d'attente. Cette démo aura deux cadres. Le premier contiendra (pour l'utilisateur) trois boutons. Un pour une file d'attente FIFO, un pour une file d'attente LIFO, et un autre pour une file de priorité. Le second cadre contiendra un widget champ de texte, deux boutons, l'un pour ajouter à la file d'attente et l'autre pour retirer de la file, et trois labels, l'un montrant quand la file est vide, l'un montrant quand la file est pleine, et un dernier pour afficher ce qui a été retiré de la file d'attente. Nous allons également écrire du code pour centrer automatiquement la fenêtre sur l'écran. Voici le début du code (encadré ci-dessus haut de la deuxième colonne).

Ici, nous avons nos importations et le début de notre classe. Comme précédemment, nous créons la routine init avec les routines DefinirVariables, Cons-

```
import sys
from Tkinter import *
import ttk
import tkMessageBox
import Queue

class TestFiles:
  def __init__(self,principale = None):
      self.DefinirVariables()
      f = self.ConstruireWidgets(principale)
      self.PlacerWidgets(f)
      self.AfficherStatut()
```

```
def DefinirVariables(self):
      self.TypeDeFile = ''
      self.StatutPlein = StringVar()
      self.StatutVide = StringVar()
      self.Element = StringVar()
      self.Sortie = StringVar()
      # Definit les files
      self.fifo = Queue.Queue(10)
      self.lifo = Queue.LifoQueue(10)
      self.pq = Queue.PriorityQueue(10)
      self.obj = self.fifo
```

```
def ConstruireWidgets(self,principale):
  # Definit nos widgets
  fenetre = Frame(principale)
  self.f1 = Frame(fenetre,
      relief = SUNKEN,
      borderwidth=2,
      width = 300,
      padx = 3,
      pady = 3
  )
  self.btnFifo = Button(self.f1,
      text = "FIFO"
  )

  self.btnFifo.bind('<Button-1>',
      lambda e: self.btnMain(1)
  )
  self.btnLifo = Button(self.f1,
      text = "LIFO"
  )
  self.btnLifo.bind('<ButtonRelease-1>',
      lambda e: self.btnMain(2)
  )
  self.btnPriority = Button(self.f1,
      text = "PRIORITY"
  )
  self.btnPriority.bind('<ButtonRelease-1>',
      lambda e: self.btnMain(3)
  )
```

truireWidgets et PlacerWidgets. Nous avons aussi une routine appelée AfficherStatut qui… affichera l'état de notre file d'attente (encadré page précédente, en haut au milieu).

Nous allons maintenant créer notre routine DefinirVariables. Nous avons quatre objets StringVar(), une variable vide appelée TypeDeFile, et trois objets file d'attente - un pour chaque type de file d'attente avec lesquels nous allons jouer. Nous avons fixé la taille maximale des files d'attente à 10 pour les besoins de la démo. Nous avons aussi créé un objet appelé obj auquel nous assignons la valeur FIFO. Lorsque nous sélectionnerons un type de file avec les boutons, nous mettrons dans cet objet le type de file d'attente que nous voulons. De cette façon, une file d'attente est conservée quand on passe à un autre type de file d'attente (encadré page précédente, en haut à droite).

Ici nous commençons la définition des widgets. Nous créons notre premier cadre, les trois boutons et leurs fonctions de rappel. Notez que nous utilisons la même routine pour gérer les fonctions de rappel. Chaque bouton envoie une valeur à la routine de rappel pour indiquer quel bouton a été cliqué. Nous aurions tout aussi bien pu créer une routine dédiée pour chaque bouton. Cependant, puisque les trois boutons gèrent une tâche commune, j'ai pensé qu'il serait bon de les considérer comme un groupe (page précédente, en bas à droite).

Ensuite nous mettons en place le second cadre, le widget de saisie et les deux boutons. La seule chose ici qui sort de l'ordinaire est le rappel pour le widget de saisie. Ici nous associons la routine self.AjouterALaFile à la touche « Return » (Entrée). De cette façon, l'utilisateur n'a pas à utiliser la souris pour ajouter les données. Il peut simplement entrer les données dans la zone de saisie et appuyer sur Entrée (encadré ci-contre, en haut).

Voici les trois dernières définitions de widgets. Toutes les trois sont des étiquettes. Nous réglons l'attribut textvariable des variables que nous avons définies plus tôt. Si vous vous souvenez, lorsque cette variable change, le texte de l'étiquette changera aussi. Nous faisons aussi quelque chose d'un peu différent sur l'étiquette lblData. Nous allons utiliser une police différente pour faire ressortir l'affichage des données extraites de la file d'attente. Rappelez-vous que nous devons retourner l'objet fenêtre de sorte qu'il puisse être utilisé dans la routine PlacerWidgets (ci-contre en bas).

C'est le début de la routine PlacerWidgets. Remarquez que nous avons mis ici cinq étiquettes vides tout en haut de la fenêtre racine. Je fais cela pour régler

```
self.f2 = Frame(fenetre,
    relief = SUNKEN,
    borderwidth=2,
    width = 300,
    padx = 3,
    pady = 3
)
self.txtAdd = Entry(self.f2,
    width=5,
    textvar=self.Element
)
self.txtAdd.bind('<Return>',self.AjouterALaFile)
self.btnAdd = Button(self.f2,
    text='Ajout dans la file',
    padx = 3,
    pady = 3
)
self.btnAdd.bind('<ButtonRelease-1>',self.AjouterALaFile)
self.btnGet = Button(self.f2,
    text='Recupere element suivant',
    padx = 3,
    pady = 3
)
self.btnGet.bind('<ButtonRelease-1>',self.RecupererDansFile)
```

```
    self.lblEmpty = Label(self.f2,
        textvariable=self.StatutVide,
        relief=FLAT
    )
    self.lblFull = Label(self.f2,
        textvariable=self.StatutPlein,
        relief=FLAT
    )
    self.lblData = Label(self.f2,
        textvariable=self.Sortie,
        relief = FLAT,
        font=("Helvetica", 16),
        padx = 5
    )

    return fenetre
```

l'espacement. C'est un moyen facile de « tricher » pour faciliter le placement de la fenêtre. Nous réglons ensuite le premier cadre, puis une autre étiquette « de triche », puis les trois boutons .

Nous plaçons maintenant le deuxième cadre, encore une étiquette « de triche » puis le reste de nos widgets.

Ensuite nous avons notre routine « standard » pour quitter l'application, qui appelle simplement sys.exit() :

```
def Quitter(self):
    sys.exit()
```

Maintenant, notre routine principale de rappel pour les boutons, btnMain. Rappelez-vous que nous lui envoyons (via le paramètre p1) quel bouton a été cliqué. Nous utilisons la variable self.TypeDeFile en référence au type de file d'attente que nous sommes en train de gérer, puis nous assignons à self.obj la file d'attente appropriée et, enfin, changeons le titre de notre fenêtre racine pour afficher le type de file d'attente que nous utilisons. Après cela, nous affichons le type de file dans le terminal (vous n'êtes pas obligé de faire cela), puis appelons la routine AfficherStatut. Maintenant nous allons écrire la routine AfficherStatut (page suivante, encadré en haut, à droite).

Comme vous pouvez le voir, c'est assez simple. Nous réglons les variables d'étiquettes à leur bon état afin qu'elles

```
def btnMain(self,p1):
    if p1 == 1:
        self.TypeDeFile = 'FIFO'
        self.obj = self.fifo
        root.title('Tests Files - FIFO')
    elif p1 == 2:
        self.TypeDeFile = 'LIFO'
        self.obj = self.lifo
        root.title('Tests Files - LIFO')
    elif p1 == 3:
        self.TypeDeFile = 'PRIORITY'
        self.obj = self.pq
        root.title('Tests Files - Priorite')
    print self.TypeDeFile
    self.AfficherStatut()
```

```
    self.f2.grid(column = 0,row = 2,sticky='nsew',columnspan=5,padx = 5, pady = 5)
    l = Label(self.f2,text='',width = 15,anchor = 'e').grid(column = 0, row = 0)
    self.txtAdd.grid(column=1,row=0)
    self.btnAdd.grid(column=2,row=0)
    self.btnGet.grid(column=3,row=0)
    self.lblEmpty.grid(column=2,row=1)
    self.lblFull.grid(column=3,row = 1)
    self.lblData.grid(column = 4,row = 0)
```

```
def PlacerWidgets(self, principale):
    fenetre = principale
    # Place les widgets
    fenetre.grid(column = 0, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 0, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 1, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 2, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 3, row = 0)
    l = Label(fenetre,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 4, row = 0)

    self.f1.grid(column = 0,row = 1,sticky='nsew',columnspan=5,padx = 5,pady = 5)
    l = Label(self.f1,text='',width = 25,anchor = 'e').grid(column = 0, row = 0)
    self.btnFifo.grid(column = 1,row = 0,padx = 4)
    self.btnLifo.grid(column = 2,row = 0,padx = 4)
    self.btnPriority.grid(column = 3, row = 0, padx = 4)
```

affichent si la file d'attente que nous utilisons est pleine, vide, ou quelque part entre les deux.

La routine AjouterALaFile est également assez simple. Nous récupérons les données du champ de saisie en utilisant la fonction .get(). Nous vérifions ensuite si le type courant de file d'attente est une file de priorité. Si c'est le cas, nous devons nous assurer que le format de saisie est correct. Nous vérifions cela en testant la présence d'une virgule. S'il n'y en a pas, nous prévenons l'utilisateur via une boîte de message d'erreur. Si tout semble correct, nous vérifions ensuite si la file d'attente que nous utilisons actuellement est pleine (encadré ci-contre, en bas). N'oubliez pas, si la file est pleine, la routine d'insertion est bloquée et le programme va planter. Si tout va bien, nous ajoutons l'élément à la file d'attente et mettons à jour le statut.

La routine RecupererDansFile est encore plus facile. Nous vérifions si la file est vide afin de ne pas nous heurter à un problème de blocage et, si ce n'est pas le cas, nous retirons les données de la file d'attente, l'affichons, et mettons à jour le statut (encadré ci-contre, au milieu).

Nous arrivons à la fin de notre application. Voici la routine de centrage de fenêtre. Nous récupérons d'abord la largeur et la hauteur de l'écran. Nous récupérons ensuite la largeur et la hauteur de la fenêtre racine à l'aide des routines winfo_reqwidth() et winfo_reqheight() intégrées à tkinter. Ces routines, lorsqu'elles sont appelées au bon moment, retourneront la largeur et la hauteur de la fenêtre racine en tenant compte du placement des widgets. Si vous l'appelez trop tôt, vous obtiendrez des valeurs, mais pas celles dont vous avez vraiment besoin. Nous soustrayons ensuite la largeur de la fenêtre de la largeur de l'écran, et divisons cela par 2, puis nous faisons la même chose pour la hauteur. Nous utilisons alors ces informations

```
if name == 'main':
  def Centrer(window):
      # recupere largeur et hauteur de l'ecran
      largeurE = window.winfo_screenwidth()
      hauteurE = window.winfo_screenheight()
      # recupere largeur et hauteur de la fenetre
      largeurF = window.winfo_reqwidth()
      hauteurF = window.winfo_reqheight()
      xc = (largeurE-largeurF)/2
      yc = (hauteurE-hauteurF)/2

window.geometry("%dx%d+%d+%d"%(largeurF,hauteurF,xc,yc))
      window.deiconify()
```

```
def AfficherStatut(self):
    # verifie si vide
    if self.obj.empty() == True:
        self.StatutVide.set('Vide')
    else:
        self.StatutVide.set('')
    # verifie si plein
    if self.obj.full() == True:
        self.StatutPlein.set('Plein')
    else:
        self.StatutPlein.set('')
```

```
def RecupererDansFile(self,p1):
    self.Sortie.set('')
    if not self.obj.empty():
        temp = self.obj.get()
        self.Sortie.set("Sorti
{0}".format(temp))
    self.AfficherStatut()
```

```
def AjouterALaFile(self,p1):
    temp = self.Element.get()
    if self.TypeDeFile == 'PRIORITY':
        commapos = temp.find(',')
        if commapos == -1:
            print "ERREUR"
            tkMessageBox.showerror('Demo File',
                'Un element Priority doit etre au
format\r(priorite,valeur)')
        else:
            self.obj.put(self.Element.get())
    elif not self.obj.full():
        self.obj.put(self.Element.get())
    self.Element.set('')
    self.AfficherStatut()
```

dans l'appel de la fonction geometry. La plupart du temps, cela fonctionne à merveille. Toutefois, il pourrait y avoir des moments où vous aurez besoin de définir la largeur et la hauteur à la main (encadré haut de la deuxième colonne, page précédente).

```
root = Tk()
root.title('Tests File - FIFO')
demo = TestFiles(root)
root.after(3,Centrer,root)
root.mainloop()
```

Enfin, nous instancions la fenêtre racine, définissons le titre de base et instancions la class TestFiles. Nous appelons ensuite root.after, qui attend un nombre x de millisecondes (dans ce cas 3), après que la fenêtre racine soit instanciée, puis appelle la routine Centrer. De cette façon, la fenêtre racine a été complètement paramétrée et est prête à s'afficher, donc nous pouvons obtenir sa largeur et sa hauteur. Vous pourriez avoir à ajuster légèrement le temps de retard. Certaines machines sont beaucoup plus rapides que d'autres. 3 fonctionne très bien sur ma machine, votre réglage peut varier. Enfin nous appelons la boucle principale de la fenêtre racine pour exécuter l'application.

Pendant que vous jouez avec les files d'attente, notez que si vous mettez des données dans une file d'attente (disons la file d'attente FIFO), puis passez à une autre file d'attente (disons la file d'attente LIFO), les données qui ont été placées dans la file FIFO sont toujours là et vous attendent. Vous pouvez complètement ou partiellement remplir les trois files d'attente, puis commencer à jouer avec.

Eh bien, c'est tout pour cette fois-ci. Amusez-vous avec vos files d'attente. Le code de TestFiles peut être trouvé ici : http://pastebin.com/MKLTmSES.

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.com.

TUTORIEL

Programmer en Python - Partie 28

Greg Walters

Nous allons approfondir l'exploration des widgets fournis par Tkinter. Cette fois, nous allons examiner les menus, listes déroulantes, les boîtes de sélection, barres de séparation, barres de progression et les onglets. Nous en parlerons à tour de rôle.

Vous avez vu des menus dans presque toutes les applications que vous utilisez. Tkinter rend très facile la création des menus. Les listes déroulantes sont similaires aux listes dont nous avons parlé dans le dernier article sur la démo des widgets, sauf que la liste se déroule vers le bas au lieu d'être visible en permanence. Les contrôles de sélection numérique sont pratiques pour définir une plage fixe de valeurs dans laquelle on peut se déplacer vers le haut ou vers le bas. Par exemple, si nous voulons que l'utilisateur soit en mesure de choisir des entiers compris entre 1 et 100, nous pouvons facilement utiliser une boîte de sélection. Les barres de progression sont une merveilleuse façon de montrer que votre application n'a pas planté quand quelque chose prend beaucoup de temps, comme la lecture des enregistrements d'une base de données. Elles peuvent montrer le pourcentage d'achèvement d'une tâche. Il y a deux types de barres de progression, déterminée et indéterminée. Vous utilisez une barre de progression déterminée quand vous savez exactement combien d'actions vous devez réaliser. Si vous ne connaissez pas le nombre d'actions ou le pourcentage de progression de votre tâche à un instant t, vous pouvez utiliser la version indéterminée. Nous allons travailler avec les deux. Enfin, un widget à onglets verticaux (ou widget à onglets horizontaux) est régulièrement utilisé pour les réglages des écrans de configuration. Vous pouvez regrouper logiquement une série de widgets sur chaque onglet.

Nous allons donc commencer. Comme d'habitude, nous allons créer une application de base et construire notre programme avec des widgets supplémentaires, que nous allons lui ajouter. Regardez à droite pour la première partie de notre application. Vous avez déjà vu presque tout cela.

Enregistrez le code en tant que widgetdemo2a.py. Rappelez-vous, nous allons l'utiliser comme base pour construire la démonstration complète. Maintenant, nous allons commencer le processus de création du menu. Voici les étapes que nous allons suivre. Premièrement, nous définissons une variable pour contenir l'occurrence de menu. Comme la plupart des widgets que nous utilisons, le format est :

```
NotreVariable =
Widget(parent, options).
```

```
import sys
from Tkinter import *
import ttk
# Montre comment créer un menu
class WidgetDemo2:

    def __init__(self,principale = None):
        self.DefinirVariables()
        f = self.ConstruireWidgets(principale)
        self.PlacerWidgets(f)

    def DefineVars(self):
        pass
```

Et voici la fin de notre programme. Vous avez déjà vu ça précédemment, rien de nouveau ici.

```
if __name__ == '__main__':
    def Center(window):
      # recupere largeur et hauteur de l'ecran
      sw = window.winfo_screenwidth()
      sh = window.winfo_screenheight()
      # recupere largeur et hauteur de la fenetre
      rw = window.winfo_reqwidth()
      rh = window.winfo_reqheight()
      xc = (sw-rw)/2
      yc = (sh-rh)/2
      print "{0}x{1}".format(rw,rh)
        window.geometry("%dx%d+%d+%d"%(rw,rh,xc,yc))
        window.deiconify()

  root = Tk()
  root.title('Demo de plus de widgets')
  demo = DemoWidget2(root)
  root.after(13,Center,root)
  root.mainloop()
```

Dans le cas présent, nous utilisons le widget Menu avec l'attribut « principale » en tant que fenêtre-mère. Nous faisons cela dans la routine ConstruireWidgets. Ensuite, nous créons un autre élément de menu, cette fois-ci en le nommant menuFichier. Nous ajouterons des commandes et des séparateurs, au besoin. Enfin, nous l'ajoutons à la barre de menu et continuons de la sorte jusqu'à ce que nous ayons fini. Dans notre exemple, nous allons avoir la barre de menu, un menu déroulant Fichier, un menu déroulant Edition et un menu déroulant Aide (en haut à droite). Commençons.

Ensuite (au milieu à droite), nous nous concentrons sur le menu Fichier. Il contiendra cinq éléments. Nouveau, Ouvrir, Sauver, un séparateur et Quitter. Nous allons utiliser la méthode .add_command pour ajouter les commandes. Tout ce que nous devons faire, c'est appeler la méthode avec le texte (label =) et ensuite fournir une fonction de rappel pour prendre la main quand l'utilisateur clique sur l'élément. Enfin, nous utilisons la fonction menubar.add_cascade pour attacher le menu à la barre.

Notez que la commande Quitter utilise « root.quit » pour mettre fin au programme. Pas besoin de fonction de rappel pour cela. Ensuite, nous ferons la même chose pour les menus Edition et Aide.

Notez la partie « tearoff = 0 » dans chacune des définitions de groupe de menu. Si vous changez le 0 en 1, le menu commencera par une sorte de ligne pointillée qui permet de détacher le menu de la barre de menus en créant sa propre fenêtre. Bien que cela puisse être utile dans le futur, ce n'est pas ce que nous voulons ici.

Enfin et surtout, nous devons placer le menu. Nous ne faisons pas un placement normal avec la fonction .grid(). Nous allons simplement l'ajouter en utilisant la fonction parent.config (en bas à droite).

Tout cela est placé dans la routine ConstruireWidgets. Maintenant (page suivante, en haut à droite), nous avons besoin d'ajouter un cadre générique et de mettre l'instruction de retour avant de passer à la routine PlacerWidgets.

Enfin (page suivante, en bas), nous devons créer toutes les fonctions de rappel que nous avons définies plus tôt. Pour la démo, nous allons simplement afficher quelque chose dans le terminal utilisé pour lancer le programme.

C'est tout. Enregistrez et exécutez le programme. Cliquez sur chacune des options de menu (en gardant Fichier|Quitter pour la fin).

```
def ConstruireWidgets(self,principale):
  fenetre = Frame(principale)
    #==============================
    #       LES MENUS
    #==============================
    # Creation de la barre de menus
  self.barreMenus = Menu(principale)
```

```
# Creation du menu Fichier et ajout a la barre de menus
menuFichier = Menu(self.barreMenus, tearoff = 0)
menuFichier.add_command(label = "Nouveau", command = self.FichierNouveau)
menuFichier.add_command(label = "Ouvrir", command = self.FichierOuvrir)
menuFichier.add_command(label = "Sauver", command = self.FichierSauver)
menuFichier.add_separator()
menuFichier.add_command(label = "Quitter", command = root.quit)
self.barreMenus.add_cascade(label = "Fichier", menu = menuFichier)
```

```
 # Creation du menu Edition
menuEdition = Menu(self.barreMenus, tearoff = 0)
menuEdition.add_command(label = "Couper", command = self.EditionCouper)
menuEdition.add_command(label = "Copier", command = self.EditionCopier)
menuEdition.add_command(label = "Coller", command = self.EditionColler)
self.barreMenus.add_cascade(label = "Edition", menu = menuEdition)
# Creation du menu Aide
menuAide = Menu(self.barreMenus, tearoff=0)
menuAide.add_command(label = "A propos", command = self.AideApropos)
self.barreMenus.add_cascade(label = "Aide", menu = menuAide)
```

```
  # affichage du menu
 principale.config(menu = self.barreMenus)
  #=======================================
  #           FIN DES MENUS
  #=======================================
```

Maintenant (ci-dessous), nous allons traiter la liste déroulante. Enregistrez votre fichier sous widgetdemo2b.py et nous serons prêts à commencer. Les importations, les définitions de classes et la routine __init__ sont toutes les mêmes, ainsi que la partie inférieure du programme. Nous allons ajouter deux lignes à la routine DefinirVariables. Commentez ou effacez l'instruction « pass » et mettez le code suivant (j'ai inclus la ligne de définition juste pour la clarté).

Nous définissons d'abord une étiquette, comme nous l'avons déjà fait. Ensuite, nous définissons la liste déroulante. Nous utilisons « ttk.Combobox », définissons le parent et réglons la hauteur à 19, la largeur à 20 et le textvariable à « self.selectionListeDeroulante1 ». Rappelez-vous que nous avons utilisé les « textvariables » dans le dernier article, mais juste au cas où vous l'auriez oublié… il change à tout moment sa valeur dès que la liste déroulante est modifiée. Nous l'avons défini dans DefinirVariables comme un objet StringVar. Ensuite nous chargeons les valeurs que nous voulons que l'utilisateur puisse choisir, et de nouveau, nous les avons définies dans DefinirVariables. Enfin, nous lions l'événement virtuel « ComboboxSelected » à la routine testListeDeroulante que nous allons étoffer dans une minute.

Ensuite, nous allons placer la liste déroulante et le titre dans notre fenêtre (page suivante en haut à droite).

```
        self.f1 = Frame(fenetre,
                        relief = SUNKEN,
                        borderwidth = 2,
                        width = 500,
                        height = 100
                        )

        return fenetre
```

Ensuite (comme nous l'avons déjà fait) nous nous occupons de placer les autres widgets.

```
    def PlacerWidgets(self,principale):
      fenetre = principale
      fenetre.grid(column = 0, row = 0)

      self.f1.grid(column = 0,
                   row = 0,
                   sticky = 'nsew'
                   )
```

```
    def DefinirVariables(self):
      self.selectionListeDeroulante1 = StringVar()
      self.valeursC1 = ['Neant','Option 1','Option 2','Option 3']
```

Insérez le code suivant dans ConstruireWidgets après la définition self.f1 et avant la ligne « return fenetre ».

```
      self.labelListeDeroulante = Label(self.f1, text = "Liste deroulante : ")
      self.listeDeroulante1 = ttk.Combobox(self.f1,
                              height = "19",
                              width = "20",
                              textvariable = self.selectionListeDeroulante1
                             )
      self.listeDeroulante1['values'] = self.valeursC1
      # associe l'evenement virtuel a une fonction de rappel

self.listeDeroulante1.bind("<<ComboboxSelected>>",self.testListeDeroulante)
```

```
def FichierNouveau(self):
    print "Menu - Fichier Nouveau"
def FichierOuvrir(self):
    print "Menu - Fichier Ouvrir"
def FichierSauver(self):
    print "Menu - Fichier Sauver"
def EditionCouper(self):
    print "Menu - Edition Couper"
def EditionCopier(self):
    print "Menu - Edition Copier"
def EditionColler(self):
    print "Menu - Edition Coller"
def AideApropos(self):
    print "Menu - Aide Apropos"
```

Sauvegardez tout et testez.

Maintenant enregistrez sous widgetdemo2c.py et nous allons commencer avec la barre de séparation. C'est super facile. Alors que les mises à jour de Tkinter fournissent un widget barre de séparation, je n'ai jamais été en mesure de le faire fonctionner. Voici une autre façon de faire. Nous utiliserons un cadre avec une hauteur de 2. Les seuls changements à notre programme seront la définition du cadre dans ConstruireWidgets après l'instruction « bind » de la liste déroulante et le placement du cadre dans la routine PlacerWidgets. Donc dans ConstruireWidgets ajoutez les lignes suivantes (montrées au milieu à droite).

Vous avez déjà vu tout cela avant. Enregistrez et testez. Vous aurez probablement à élargir la fenêtre de premier niveau pour voir le séparateur, tout cela va être bien plus évident dans la prochaine démo. Enregistrez en tant que widgetdemo2d.py, nous allons ajouter la zone de sélection numérique.

Sous DefinirVariables, ajoutez la ligne suivante :

```
self.valeurSelection =
StringVar()
```

Maintenant, vous savez que c'est pour pouvoir obtenir sa valeur à tout moment. Ensuite, nous allons ajouter du code à la routine ConstruireWidgets, juste avant la ligne « return fenetre » (en bas à droite).

Ici, nous définissons une étiquette et la zone de sélection numérique. La définition de la zone de sélection est :

```
notreWidget =
Spinbox(parent,valeur basse,
valeur haute, largeur,
textvariable, wrap)
```

La valeur mini doit s'appeler « from_ » car le mot « from » est un mot-clé réservé du langage python et l'utiliser risquerait de casser votre programme. Les valeurs « from_ » et « to » doivent être définies comme valeurs flottantes. Ici, nous voulons que la valeur mini soit 1 et la valeur maxi soit 10. Enfin l'option « wrap » signifie que si la valeur est (dans notre cas) 10 et que l'utilisateur clique sur la flèche du haut, nous voulons qu'il retourne à la valeur mini et ainsi de suite. Il en est de même pour la valeur mini. Si l'utilisateur clique sur la flèche du bas alors que la valeur est 1, il retourne à 10 et ainsi de suite. Si vous mettez « wrap = False », le mécanisme s'arrête simplement et il n'y a pas de bouclage.

Maintenant, nous allons placer les widgets dans PlacerWidgets (page suivante en bas à gauche).

Voilà, c'est tout. Enregistrez et jouez. Vous voyez nettement le séparateur maintenant. Enregistrez en tant que widgetdemo2e.py et nous allons créer les barres de progression.

```
self.labelSelection.grid(column = 0, row = 4)
self.selection1.grid(column = 1,
                     row = 4,
                     pady = 2
                       )
```

Et enfin on écrit la fonction de retour qui affiche simplement dans le terminal ce que l'utilisateur a choisi.

```
def testListeDeroulante(self,p1):
print self.selectionListeDeroulante1.get()
```

```
self.separ = Frame(self.f1,
                   width = 140,
                   height = 2,
                   relief = RIDGE,
                   borderwidth = 2
                   )
```

Puis ajoutez cela dans PlacerWidgets…

```
self.separ.grid(column = 0,
                row = 3,
                columnspan = 8,
                sticky = 'we',
                padx = 3,
                pady = 3
                )
```

```
self.labelSelection = Label(self.f1, text = "Selection numerique :")
self.selection1 = Spinbox(self.f1,
                          from_ = 1.0,
                          to = 10.0,
                          width = 3,
                          textvariable = self.valeurSelection,
                          wrap=True
                          )
```

Encore une fois, nous avons besoin de définir certaines variables, dans la routine DefinirVariables ajoutez le code suivant :

```
 self.valeurSelection2 =
StringVar ()
self.boutonEtat = False
self.valeurBarreProg2 =
StringVar ()
```

Il est assez évident de deviner ce que sont les deux variables StringVar. Nous parlerons de self.boutonEtat dans un instant. Pour le moment, continuons et définissons les widgets pour cette portion dans Construire-Widgets (à droite).

De nouveau ceci est placé avant le « return fenetre ». Ce que nous faisons c'est la mise en place d'un cadre qui contiendra les widgets. Puis, nous avons mis en place deux étiquettes comme guides. Et nous définissons la première barre de progression. Les seules choses qui pourraient être étranges sont la longueur, le mode et le maximum. La longueur est la taille en pixels de notre barre. Le maximum est la valeur la plus élevée possible. Dans ce cas, c'est 100 comme nous utilisons des pourcentages. Dans le cas présent mode vaut « indéterminé ». Rappelez-vous, nous utilisons ce mode lorsque nous ne savons pas précisément où nous en sommes dans la progression d'une tâche, mais que nous voulons que l'utilisateur sache qu'il se passe toujours quelque chose.

Maintenant, nous ajoutons un bouton (vous l'avez déjà fait), une autre étiquette, une autre barre de progression et une autre zone de sélection numérique. Le mode de cette seconde barre de progression est « déterminé ». Nous utilisons la zone de sélection numérique pour régler le « pourcentage » d'achèvement. Puis, ajoutons les lignes suivantes (page suivante, en haut à droite) dans la routine PlacerWidgets.

Finalement, nous ajoutons deux routines pour contrôler nos barres de progression (page suivante en bas à droite).

La routine TestBarreProg contrôle la barre de progression indéterminée. Simplement, nous démarrons et arrêtons une horloge interne qui est intégrée dans la barre de progression. La ligne « self.barreProg.start(10) » paramètre le minuteur à 10 millisecondes. Cela rend le mouvement de la barre assez rapide. N'hésitez pas à jouer avec cette valeur à la hausse ou à la baisse. La routine SelectionAction définit simplement l'avancement

```
self.labelSelection.grid(column = 0, row = 4)
self.selection1.grid(column = 1,
                     row = 4,
                     pady = 2
                     )
```

```
  #=======================================
  #        BARRE DE PROGRESSION
  #=======================================
self.lbl0 = Label(self.fBarreProg,
                  text = "Barres de progression"
                  )
self.lbl1 = Label(self.fBarreProg,
                  text = "Indeterminee",
                  anchor = 'e'
                  )
self.barreProg = ttk.Progressbar(self.fBarreProg,
                  orient = HORIZONTAL,
                  length = 100,
                  mode = 'indeterminate',
                  maximum = 100
                  )
self.btnptest = Button(self.fBarreProg,
                  text = "Demarrer",
                  command = self.TestBarreProg
                  )
self.lbl2 = Label(self.fBarreProg,
                  text = "Determinee"
                  )
self.barreProg2 = ttk.Progressbar(self.fBarreProg,
                  orient = HORIZONTAL,
                  length = 100,
                  mode = 'determinate',
                  variable = self.valeurBarreProg2
                  )
self.selection2 = Spinbox(self.fBarreProg,
                  from_ = 1.0,
                  to = 100.0,
                  textvariable = self.valeurSelection2,
                  wrap = True,
                  width = 5,
                  command = self.SelectionAction
                  )
```

de la barre de progression en fonction de la valeur sélectionnée. Nous l'affichons dans un terminal.

C'est tout pour le moment. Sauvegardez et jouez.

Maintenant sauvegardez sous le nom widgetdemo2f.py et nous allons nous occuper des onglets. Ajoutez le code suivant dans ConstruireWidgets (ci-dessous) avant la ligne « return fenetre » :

Regardons ce que nous avons fait. Premièrement, nous définissons un cadre pour notre widget « onglets ». Puis nous définissons le widget. Nous avons déjà rencontré toutes les options auparavant. Ensuite, nous définissons deux cadres nommés self.p1 et self.p2 qui seront nos pages. Les deux lignes suivantes (self.on-glets.add) attachent les cadres au widget et ils ont un onglet qui leur est rattaché. Nous avons également réglé les titres des onglets. Enfin, nous mettons une étiquette sur la page numéro un. Nous allons en mettre une sur la page deux lorsque nous plaçerons les contrôles juste pour le plaisir.

Dans la routine PlacerWidgets, insérez le code suivant (page suivante en bas à gauche).

```
        #=======================================
        #               ONGLETS
        #=======================================
        self.fenetreOnglets = Frame(self.f1,
                            relief = SUNKEN,
                            borderwidth = 2,
                            width = 500,
                            height = 300
                            )
      self.onglets = ttk.Notebook(self.fenetreOnglets,
                                  width = 490,
                                  height = 290
                                  )
      self.p1 = Frame(self.onglets)
      self.p2 = Frame(self.onglets)
      self.onglets.add(self.p1,text = 'Page 1')
      self.onglets.add(self.p2,text = 'Page 2')
      self.labelPage1 = Label(self.p1,
                          text = "Voici un texte sur la
page 1",
                          padx = 3,
                          pady = 3
                            )
```

```
  # Barre de progression
self.fBarreProg.grid(column = 0,
                     row = 5,
                     columnspan = 8,
                     sticky = 'nsew',
                     padx = 3,
                     pady = 3
                     )
self.lbl0.grid(column = 0, row = 0)
self.lbl1.grid(column = 0,
                    row = 1,
                    pady = 3
                    )
self.barreProg.grid(column = 1, row = 1)
self.btnptest.grid(column = 3, row = 1)
self.lbl2.grid(column = 0,
                    row = 2,
                    pady = 3
                    )
self.barreProg2.grid(column = 1, row = 2)
self.selection2.grid(column = 3, row = 2)
```

```
def TestBarreProg(self):
    if self.boutonEtat == False:
        self.btnptest.config(text="Arreter")
        self.boutonEtat = True
        self.barreProg.start(10)
    else:
        self.btnptest.config(text="Demarrer")
        self.boutonEtat = False
        self.barreProg.stop()

def SelectionAction(self):
    v = self.valeurSelection2.get()
    print v
    self.valeurBarreProg2.set(v)
```

La seule chose qui pourrait paraître étrange, c'est l'étiquette sur la page deux. Nous combinons la définition et la mise en place dans la grille avec la même commande. Nous l'avons déjà fait dans notre première application de démo.

C'est fini. Sauvegardez et amusez-vous.

Comme toujours le code de l'application complète est sur pastebin :
http://pastebin.com/7BJr54au.

Au plaisir. La prochaine fois nous allons plutôt aborder des trucs sur les bases de données.

```
    self.fenetreOnglets.grid(column = 0,
                             row = 6,
                             columnspan = 8,
                             rowspan = 7,
                             sticky = 'nsew'
                             )
    self.onglets.grid(column = 0,
                      row = 0,
                      columnspan = 11,
                      sticky = 'nsew'
                      )
    self.labelPage1.grid(column = 0,row = 0)
    self.labelPage2 = Label(self.p2,
                      text = 'Voici un texte sur la page 2',
                      padx = 3,
                      pady = 3
                      ).grid(
                             column = 0,
                             row = 1
                               )
```

TUTORIEL

Programmer en Python - Partie 29

Greg Walters

Il y a quelque temps, on m'a demandé de convertir une base de données MySQL en SQLite. En cherchant une solution rapide et facile (et gratuite) sur internet, je n'ai rien trouvé qui fonctionnait pour moi avec la version actuelle de MySQL. Alors j'ai décidé d'aller de l'avant et de fabriquer ma solution moi-même.

Le programme MySQL Administrator vous permet de sauvegarder une base de données dans un fichier texte à plat. Beaucoup de navigateurs SQLite vous permettent de lire un fichier SQL de définition à plat et de créer la base de données à partir de là. Cependant, il y a beaucoup de choses que MySQL supporte, mais pas SQLite. Alors ce mois-ci, nous allons écrire un programme de conversion qui lit un fichier de « dump » (sauvegarde) MySQL et crée une version SQLite.

Commençons par regarder le fichier de « dump » MySQL. Il se compose d'une section qui crée la base de données, puis des sections qui créent chaque table dans la base et insèrent des données dans ces tables, si les données sont contenues dans le fichier de « dump ». (Il existe une option pour exporter seulement les schémas des tables.) Ci-contre à droite se trouve un exemple d'une des sections de création de table.

La première chose dont nous devons nous débarrasser se trouve dans la dernière ligne. Tout ce qui suit la parenthèse fermante doit disparaître. (SQLite ne supporte pas une base de données InnoDB). De plus, SQLite ne supporte pas la ligne « PRIMARY KEY ». Dans SQLite, on règle une clé primaire en utilisant « INTEGER PRIMARY KEY AUTOINCREMENT » quand nous définissons la colonne. L'autre chose que SQLite ne supporte pas est le mot-clé « unsigned ».

Quant aux données, les déclarations « INSERT INTO » sont également non-compatibles. Le problème ici est que SQLite ne permet pas les insertions multiples dans une même déclaration. Voici un court exemple tiré du fichier de « dump » MySQL. Remarquez (à droite) que le marqueur de fin de ligne est un point-virgule.

Nous allons également ignorer toutes les lignes de commentaires et

```
DROP TABLE IF EXISTS `categoriesmain`;
CREATE TABLE `categoriesmain` (
  `idCategoriesMain` int(10) unsigned NOT NULL
auto_increment,
  `CatText` char(100) NOT NULL default '',
  PRIMARY KEY  (`idCategoriesMain`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT CHARSET=latin1;
```

```
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES
 (1,'Apéritif'),
 (2,'Snack'),
 (3,'Barbecue'),
 (4,'Gateaux'),
 (5,'Bonbons'),
 (6,'Boissons');
```

Pour rendre ceci compatible, nous devons le remplacer par plusieurs insertions séparées, comme ceci :

```
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (1,'Apéritif');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (2,'Snack');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (3,'Barbecue');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (4,'Gateaux');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (5,'Bonbons');
INSERT INTO `categoriesmain` (`idCategoriesMain`,`Cat-
Text`) VALUES (6,'Boissons');
```

les instructions « CREATE DATABASE » et « USE ». Une fois que nous aurons le fichier SQL converti, nous utiliserons un programme semblable à SQLite Database Browser qui est dans le domaine public, pour réellement créer la base de données, les tables et les données.

Commençons. Ouvrez un dossier pour ce nouveau projet et un nouveau fichier python. Nommez-le MonSQLversSQLite.py.

Vous voyez en haut à droite la déclaration d'importation, la définition de classe et la routine __init__.

Ce programme sera exécuté en ligne de commande, nous avons donc besoin de créer la déclaration « if __name__ », un gestionnaire pour les arguments de ligne de commande et une routine d'utilisation (si l'utilisateur ne sait pas comment utiliser le programme). Tout cela va à la toute fin du programme. Tout le reste du code se trouvera avant ceci :

```
def error(message):
    print >> sys.stderr,
str(message)
```

Ensuite se trouve le gestionnaire qui affiche les instructions d'utilisation du programme.

La routine FaitLe() est appelée si notre programme est lancé à partir de la ligne de commande, ce pourquoi il est conçu. Cependant, si nous voulons pouvoir en faire une bibliothèque qui sera incluse dans un autre programme à un autre moment, nous pouvons simplement utiliser la classe. Ici nous avons mis en place un certain nombre de variables pour s'assurer que tout fonctionne correctement. Le code visible sur la page précédente

```
def FaitLe():
    #=====================================
    #            Variables
    #=====================================
    FichierSource = ''
    FichierDestination = ''
    Debug = False
    Aide = False
    SchemaSeulement = False
    #=====================================
```

```
    #!/usr/bin/env python
    #=====================================
    # MonSQLversSQLite.py
    #=====================================
    #            IMPORTS
    import sys
    #=====================================

    #=====================================
    #     DEBUT CLASS MonSQL2versSQLite
    #=====================================
    class MonSQLversSQLite:
        def __init__(self):
            self.FichierSource = ""
            self.FichierDestination = ""
            self.EcrireFichier = 0
            self.ModeDebug = 0
            self.SchemaSeulement = 0
            self.ModeDirect = False
```

```
if len(sys.argv) == 1:
    usage()
else:
    for a in sys.argv:
        print a
        if a.startswith("FicEntree="):
            pos = a.find("=")
            FichierSource = a[pos+1:]
        elif a.startswith("FicSortie="):
            pos = a.find("=")
            FichierDestination = a[pos+1:]
        elif a == 'Debug':
            Debug = True
        elif a == 'SchemaSeukelent':
            SchemaSeulement = True
        elif a == '-Aide' or a == '-H' or a == '-?':
            Aide = True
    if Aide == True:
        usage()
    r = MonSQLversSQLite()
    r.Init(FichierSource,FichierDest,Debug,SchemaSeul)
    r.ExecuterTravail()
```

en bas à droite analyse ensuite les arguments de ligne de commande passés à notre programme et prépare les choses pour les routines principales.

Quand nous commençons le programme, nous devons fournir au moins deux variables sur la ligne de commande. Ce sont les fichiers d'entrée et de sortie. Nous fournirons également une information pour permettre à l'utilisateur de voir ce qui se passe pendant que le programme est lancé, une option pour simplement créer les tables, ne pas charger les données et un moyen pour l'utilisateur d'appeler au secours. La ligne de commande « normale » pour démarrer le programme ressemble à ceci :

```
MonSQLversSQLite FicEntree=Foo
FicSortie=Bar
```

où « Foo » est le nom du fichier de dump MySQL, et « Bar » est le nom du fichier SQLite que le programme doit créer.

Vous pouvez également l'appeler ainsi :

```
MonSQLversSQLite FicEntree=Foo
FicSortie=Bar Debug SchemaSeu-
lement
```

```
def usage():
    message = (
       '================================================================\n'
        'MonSQLversSQLite - Un convertisseur de bases de donnees\n'
        'Auteur: Greg Walters\n'
        'USAGE:\n'
        'MonSQLversSQLite FicEntree=nomFichier [FicSortie=nomFichier] [SchemaSeulement]
[Debug] [-H-Aide-?]\n'
        '   avec\n'
        '         FicEntree est le fichier de dump MySQL\n'
        '         FicSortie (optionnel) est le nom du fichier de sortie\n'
        '            (si FicSortie est omis, on suppose une sortie directe vers SQLite)\n'
        '         SchemaSeulement (optionnel) Cree les tables, N\'IMPORTE PAS LES
DONNEES\n'
        '         Debug (optionnel) - Affiche les messages de debug\n'
        '         -H or -Aide or -? - Affiche ce message\n'
        'Copyright (C) 2011 par G.D. Walters\n'
        '================================================================\n'
        )
    error(message)
    sys.exit(1)

if __name__ == "__main__":
    FaitLe()
```

ce qui ajoutera l'option pour afficher les messages de débogage et pour créer SEULEMENT les tables sans importer les données.

Finalement si l'utilisateur demande de l'aide, on va simplement dans la section « Utilisation » du programme.

Avant de continuer, regardons à nouveau comment fonctionne la prise en charge des arguments de la ligne de commande.

Lorsqu'un utilisateur entre le nom du programme en ligne de commande (terminal), le système d'exploitation conserve la trace des informations saisies et il les passe au programme juste au cas où des options ont été saisies. Si aucune option (autrement nommée argument) n'est saisie, le nombre d'arguments est un, ce qui correspond au nom de l'application - dans notre cas, MonSQLversSQLite.py. On accède à ces arguments avec la commande sys.arg. Si le nombre est supérieur à un, nous allons utiliser une boucle for pour y accéder. Nous allons parcourir la liste des arguments et vérifier chacun d'eux. Certains programmes exigent que vous entriez les arguments dans un ordre précis. En utilisant l'approche avec une boucle for, les arguments peuvent être saisis dans n'importe quel ordre. Si l'utilisateur ne fournit pas d'argument, ou utilise l'un des arguments d'aide, on affiche l'écran d'utilisation. Ci-dessus se trouve la routine pour cela.

Ensuite, une fois que nous avons analysé l'ensemble des arguments, nous instancions la classe, appelons la rou-

tine de configuration qui remplit certaines variables et ensuite appelons la routine ExecuterTravail. Nous allons commencer notre classe maintenant (voir en bas à droite).

Voici (en haut à droite) les configurations et la routine __init__. Ici, nous configurons les variables dont nous aurons besoin tout au long du code. Souvenez-vous que juste avant d'appeler la routine ExecuterTravail, nous appelons la routine de configuration, où nous prendrons les variables vides pour leur assigner des valeurs correctes. Notez qu'on laisse la possibilité de ne pas écrire dans un fichier, ce qui est utile pour le débogage. Nous avons également la possibilité de simplement écrire le schéma (la structure de la base de données), sans écrire les données. Ceci est utile si vous prenez une base de données et commencez un nouveau projet sans vouloir utiliser des données existantes.

Nous commençons par ouvrir le fichier de « dump » SQL, puis nous définissons les variables à portée interne. Nous définissons aussi certaines chaînes pour nous éviter de les saisir plus tard.

```
    def Init(self, Entree, Sortie = '', Debug = False, Schema =
0):
        self.FichierSource = Entree
        if Sortie == '':
            self.EcrireFichier = 0
        else:
            self.EcrireFichier = 1
            self.FichierDestination = Sortie
        if Debug == True:
            self.ModeDebug = 1
        if Schema == 1:
            self.SchemaSeulement = 1
```

Maintenant passons à la routine ExecuterTravail, où la magie opère vraiment.

```
    def ExecuterTravail(self):
        f = open(self.FichierSource)
        print "Debut du processus"
        cntr = 0
        ModeInsertion = 0
        ModeCreationTable = 0
        DebutInsertion = "INSERT INTO "
        AI = "auto_increment"
        PK = "PRIMARY KEY "
        IPK = " INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL"
        CT = "CREATE TABLE "
        # Debut
        if self.EcrireFichier == 1:
            FichierDest = open(self.FichierDestination,'w')
```

```
    while 1:
        ligne = f.readline()
        cntr += 1
        if not ligne:
            break
        # Ignore les lignes vides, ou qui commencent
par "--", ou les commentaires (/*!)
        if ligne.startswith("--"): # Commentaire
            pass
        elif len(ligne) == 1: # Ligne vide
            pass
        elif ligne.startswith("/*!"): # Commentaire
            pass
        elif ligne.startswith("USE"):
            #Ignore les lignes USE
            pass
        elif ligne.startswith("CREATE DATABASE "):
            pass
```

```
#====================================
#     DEBUT CLASS MonSQLversSQLite
#====================================
class MonSQLversSQLite:
    def __init__(self):
        self.FichierSource = ""
        self.FichierDestination = ""
        self.EcrireFichier = 0
        self.ModeDebug = 0
        self.SchemaSeulement = 0
```

Ensuite, si nous prévoyons d'écrire un fichier de sortie, nous l'ouvrons puis nous commençons le processus complet. Nous lirons chaque ligne du fichier d'entrée, pour les traiter et, éventuellement, les écrire dans le fichier de sortie. Nous utilisons une boucle while infinie pour la lecture des lignes, avec une commande « break » quand il ne reste rien dans le fichier d'entrée. Nous utilisons f.readline() pour obtenir la ligne à convertir et nous l'assignons à la variable « ligne ». Certaines lignes peuvent être ignorées. Nous allons simplement utiliser une instruction if/elif suivie par une instruction pass pour cela (page précédente en bas à gauche).

Ensuite nous pouvons cesser d'ignorer les choses et agir pour de bon. Si nous rencontrons une déclaration de création de table, nous allons commencer ce processus. Rappelez-vous, nous avons défini CT comme étant égal à “CREATE TABLE”. Ici (en haut à droite), nous réglons une variable « ModeCreationTable » sur 1, pour savoir que c'est ce que nous faisons, car chaque définition de champ est sur une ligne distincte. Nous prenons ensuite notre ligne, supprimons le retour chariot, la préparons pour être écrite dans le fichier de sortie, et, si nécessaire, nous l'écrivons.

Maintenant (à droite au milieu), nous devons commencer à traiter chaque ligne contenue dans l'instruction de création de table - en manipulant chaque ligne pour que SQLite soit content. Il y a plusieurs choses que SQLite ne traitera pas. Regardons à nouveau une instruction CREATE TABLE de MySQL.

Une chose qui va vraiment poser problème à SQLite est la toute dernière ligne après la parenthèse fermante. Une autre est la ligne juste au-dessus, la ligne de clé primaire. Une autre chose est le mot-clé unsigned à la deuxième ligne. Cela va nécessiter un peu de code (en bas à gauche) pour contourner ces problèmes, mais nous pouvons y arriver.

Tout d'abord, (troisième cadre sur la droite), nous vérifions si la ligne commence par « auto_increment » (AI). Nous supposerons que ce sera la ligne de clé primaire. Bien que cela soit vrai 98,6 % du temps, ce n'est pas toujours le cas.

```
    elif ligne.startswith(CT):
        ModeCreationTable = 1
        l1 = len(ligne)
        ligne = ligne[:l1-1]
        if self.ModeDebug == 1:
            print "Debut de CREATE TABLE"
            print ligne
        if self.EcrireFichier == 1:
            FichierDest.write(ligne)
```

```
CREATE TABLE `categoriesmain` (
  `idCategoriesMain` int(10) unsigned NOT NULL auto_increment,
  `CatText` char(100) NOT NULL default '',
  PRIMARY KEY  (`idCategoriesMain`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT CHARSET=latin1;
```

```
            p1 = ligne.find(AI)
            if ligne.startswith(") "):
                ModeCreationTable = 0
                if self.ModeDebug == 1:
                    print "Fin du CREATE TABLE"
                nouvelleLigne = ");\n"
                if self.EcrireFichier == 1:
                    FichierDest.write(nouvelleLigne)
                    if self.ModeDebug == 1:
                        print "Ecriture de la ligne
{0}".format(nouvelleLigne)
```

```
            elif p1 != -1:
                # on a une ligne de cle primaire
                l = ligne.strip()
                fnpos = l.find(" int(")
                if fnpos != -1:
                    fn = l[:fnpos]
                nouvelleLigne = fn + IPK #+ ",\n"
                if self.EcrireFichier == 1:
                    FichierDest.write(nouvelleLigne)
                    if self.ModeDebug == 1:
                        print "Ecriture de la ligne
{0}".format(nouvelleLigne)
```

```
    elif ModeCreationTable == 1:
        # traite la ligne...
        if self.ModeDebug == 1:
            print "Ligne a traiter — {0}".format(ligne)
```

Cependant, nous allons garder les choses simples. Ensuite nous vérifions si la ligne commence par «) ». Cela signifie que ceci est la dernière ligne de la section CREATE TABLE. Si oui, nous écrivons simplement une chaîne pour fermer correctement la déclaration dans la variable « nouvelleLigne », réglons la variable ModeCreationTable à 0 et, si nous écrivons dans un fichier, nous réalisons l'écriture.

Maintenant (page précédente en bas à droite), nous utilisons les informations que nous avons trouvées sur le mot-clé auto_increment. Tout d'abord, nous enlevons de la ligne tous les espaces parasites, puis vérifions pour voir où se trouve (nous supposons qu'elle est là) l'expression « int(» dans la ligne. Nous la remplacerons par l'expression « INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ». La longueur de l'entier n'est pas importante pour SQLite. Encore une fois, nous l'écrivons si c'est nécessaire.

Maintenant nous cherchons l'expression « PRIMARY KEY » dans la ligne. Remarquez l'espace supplémentaire à la fin : c'est exprès. Si on la trouve, on ignore la ligne.

```
elif
ligne.strip().startswith(PK):
    pass
```

Maintenant (en haut à droite) nous recherchons l'expression « unsigned » (encore une fois remarquez les espaces supplémentaires) et la remplaçons par « ».

C'est la fin de la routine de création de table. Maintenant (ci-dessous), nous passons aux requêtes d'insertion pour les données. La variable DebutInsertion contient l'expression « INSERT INTO ». Nous vérifions cela parce que MySQL permet d'insérer plusieurs déclarations en une seule commande, mais pas SQLite. Nous devons faire des déclarations distinctes pour chaque bloc de données. Nous réglons une va-

```
        elif ligne.startswith(DebutInsertion):
            if ModeInsertion == 0:
                ModeInsertion = 1
                # recupere le nom de la table et la liste
des champs
                istatement = ligne
                # enleve le retour chariot de la ligne
istatement
                    l = len(istatement)
                    istatement = istatement[:l-2]
```

```
            elif ligne.find(" unsigned ") != -1:
                ligne = ligne.replace(" unsigned "," ")
                ligne = ligne.strip()
                longueur1 = len(ligne)
                ligne = ligne[:l1-1]
                if self.EcrireFichier == 1:
                    FichierDest.write("," + ligne)
                    if self.ModeDebug == 1:
                        print "Ecriture de la ligne
{0}".format(ligne)
```

Sinon, on peut s'occuper de la ligne.

```
            else:
                longueur1 = len(ligne)
                ligne = ligne.strip()
                ligne = ligne[:l1-4]
                if self.ModeDebug == 1:
                    print "," + ligne
                if self.EcrireFichier == 1:
                    FichierDest.write("," + ligne)
```

```
                if posx != -1:
                    longueur1 = ligne[:posx+3]
                    ModeInsertion  = 0
                    if self.ModeDebug == 1:
                        print istatement + longueur1
                        print "--------------------------"
                    if self.EcrireFichier == 1:
                        FichierDest.write(istatement +
longueur1+"\n")
```

Sinon, on concatène le prélude à l'instruction et on termine par un point-virgule.

```
                elif pos1 != -1:
                    l1 = line[:pos1+2]
                    if self.DebugMode == 1:
                        print istatement + l1 + ";"
                    if self.WriteFile == 1:
                        OutFile.write(istatement + l1 +
";\n")
```

riable appelée « ModeInsertion » à 1, plaçons le « INSERT INTO {table} {liste des champs} VALUES (» dans une variable réutilisable (que je vais appeler notre prélude), et continuons.

Maintenant, nous vérifions si nous devons seulement travailler sur le schéma. Si oui, nous pouvons ignorer sans problème toutes les instructions d'insertion. Sinon, nous devons nous en occuper.

```
elif self.SchemaSeulement == 0:
    if ModeInsertion == 1:
```

Nous vérifions s'il y a soit « '); » soit « '), » dans notre ligne. Le cas « '); » indique que c'est la dernière ligne de l'ensemble d'instructions d'insertion.

```
posx = ligne.find("');")
pos1 = ligne.find("'),")
longueur1 = ligne[:pos1]
```

Cette ligne vérifie s'il y a des apostrophes échappées et les remplace.

```
ligne = ligne.replace
("\\'","''")
```

Si nous avons une déclaration de clôture (");"), c'est alors la fin de notre ensemble d'insertions et nous pouvons créer l'instruction en concaténant le prélude à l'instruction de valeur proprement dite. Ceci est illustré en bas à droite de la page précédente.

Tout cela fonctionne (ci-dessous) si la dernière valeur que nous avons dans l'instruction INSERT est une chaîne entre guillemets. Cependant, si la dernière valeur est une valeur numérique, nous devons procéder un peu différemment. Vous serez en mesure de comprendre ce que nous faisons ici.

Enfin, nous fermons notre fichier d'entrée et, si nous écrivons un fichier de sortie, nous le fermons aussi.

```
f.close()
if self.EcrireFichier == 1:
    FichierDest.close()
```

Une fois que vous avez votre fichier converti, vous pouvez utiliser SQLite Database Browser pour remplir la structure de la base et les données.

Ce code devrait fonctionner tel quel dans plus de 90 % des cas. Nous pourrions avoir oublié certaines choses à cause d'autres problèmes, c'est pour cela qu'un mode Debug est prévu. Cependant, j'ai testé cela sur plusieurs fichiers et n'ai eu aucun problème.

Comme toujours, le code est disponible sur Pastebin : http://pastebin.com/Bdt64VqS.

À la prochaine fois.

```
else:
    if self.ModeDebug == 1:
        print "Test de la ligne {0}".format(ligne)
    pos1 = ligne.find("),")
    posx = ligne.find(");")
    if self.ModeDebug == 1:
        print "pos1 = {0}, posx = {1}".format(pos1,posx)
    if pos1 != -1:
        longueur1 = ligne[:pos1+1]
        if self.ModeDebug == 1:
            print istatement + longueur1 + ";"
        if self.EcrireFichier == 1:
            FichierDest.write(istatement + longueur1 + ";\n")
    else:
        ModeInsertion = 0
        longueur1 = ligne[:posx+1]
        if self.ModeDebug == 1:
            print istatement + longueur1 + ";"
        if self.EcrireFichier == 1:
            FichierDest.write(istatement + longueur1 + ";\n")
```

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.com.

TUTORIEL

Greg Walters

Débuter Python - Partie 30

Ce mois-ci, nous allons explorer encore un autre concepteur graphique, cette fois c'est pour Tkinter. Beaucoup de gens ont un problème avec Tkinter, car il n'offre pas un designer intégré. Alors que je vous ai montré comment concevoir facilement vos applications sans concepteur, nous allons en examiner un maintenant. Il s'appelle Page. Fondamentalement, il s'agit d'une version de Visual TCL avec une couche de Python par dessus. La version actuelle est la 3.2 et peut être trouvée à http://sourceforge.net/projects/page/files/latest/download.

Pré-requis

Vous devez avoir TCK/TK 8.5.4 ou plus, Python 2.6 ou plus et pyttk, que vous pouvez obtenir (si vous ne l'avez pas encore) à partir de http://pypi.python.org/pypi/pyttk. Vous avez probablement tous ceux-ci à l'exception possible de pyttk.

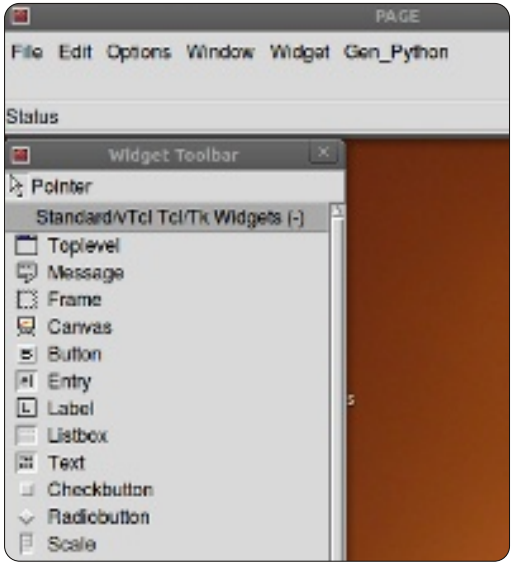
Installation

Vous ne pouvez vraiment pas demander une procédure d'installation plus facile. Il suffit de décompresser le fichier de distribution dans un dossier de votre choix. Exécutez le script appelé « configure » à partir du dossier où vous venez de tout déballer. Cela va créer votre script de lancement appelé « page » que vous utiliserez pour obtenir tout le reste. C'est tout.

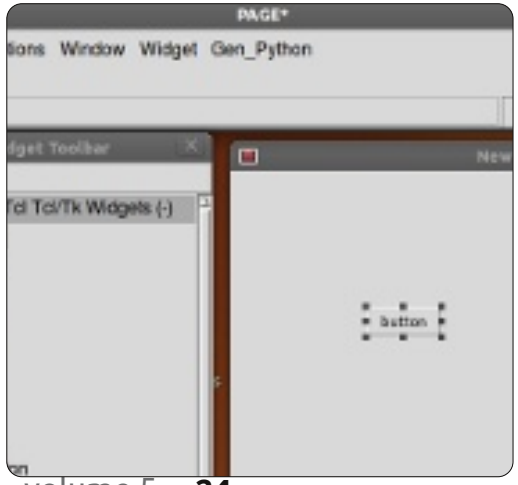
Apprentissage de page

Lorsque vous démarrez Page, vous aurez trois fenêtres (formulaires). L'une est une « piste de lancement », l'autre est une boîte à outils et la dernière montre l'éditeur d'attributs.

Pour démarrer un nouveau projet, cliquez sur le bouton du haut dans la boîte à outils.



Cela crée votre formulaire principal. Vous pouvez le déplacer où vous le souhaitez sur votre écran. Ensuite, et à partir de maintenant, cliquez sur un widget dans la boîte à outils, puis cliquez sur l'endroit où vous le voulez sur le formulaire principal.

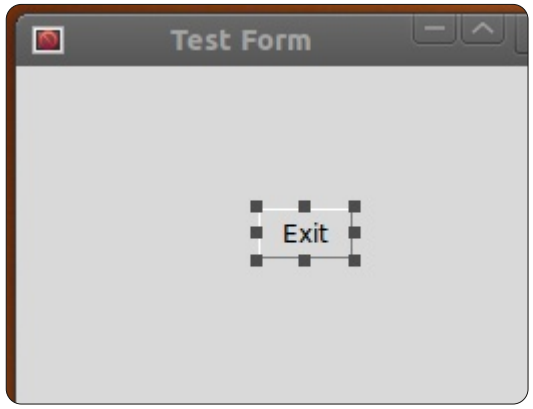


Pour l'instant, nous allons faire un bouton. Cliquez sur le bouton Button dans la boîte à outils, puis cliquez quelque part sur le formulaire principal.

Ensuite, dans le formulaire de lancement, cliquez sur Fenêtre (Window) et sélectionnez l'Éditeur d'attributs (s'il n'est pas déjà affiché). Votre bouton unique devrait être déjà mis en surbrillance ; déplacez-le dans le formulaire et lorsque vous relâchez le bouton de la souris, vous devrez voir le changement de position dans le formulaire éditeur d'attributs sous « x position » et « y position ».

Ici, nous pouvons définir d'autres attributs tels que le texte sur le bouton (ou la plupart des autres widgets), l'alias pour le widget (le nom auquel nous allons nous référer dans notre code), la couleur, le nom par lequel nous l'appellerons et plus. Près du bas de l'éditeur d'attributs se trouve le champ de texte. Ceci est le texte que l'utilisateur voit pour, dans ce cas, le widget bouton. Changeons-le de « Button » à « Exit ». Remarquez qu'à présent le bouton affiche « Exit ». Maintenant redimensionnez le formu-

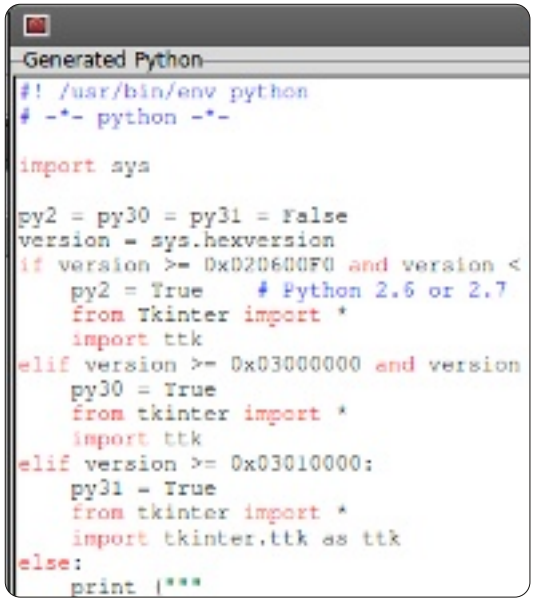
laire pour montrer seulement le bouton et recentrez le bouton dans le formulaire.



Ensuite, cliquez quelque part dans le formulaire principal où le bouton n'est pas. Le formulaire éditeur d'attributs affiche maintenant les attributs du formulaire principal. Trouvez le champ « Title » et changez-le de « New Toplevel 1 » à « Test Form ».

Maintenant, avant de sauvegarder notre projet, nous avons besoin de créer un dossier pour contenir nos fichiers de projet. Créez un dossier quelque part sur votre disque appelé « PageProjects ». Puis, dans la fenêtre de lancement, sélectionnez File puis Save As. Allez dans votre dossier PageProjects et, dans la boîte de dialogue, tapez TestForm.tclet et cliquez sur le bouton Save. Notez que c'est enregistré comme fichier TCL, pas comme fichier Python. Ensuite, nous allons créer le fichier python.

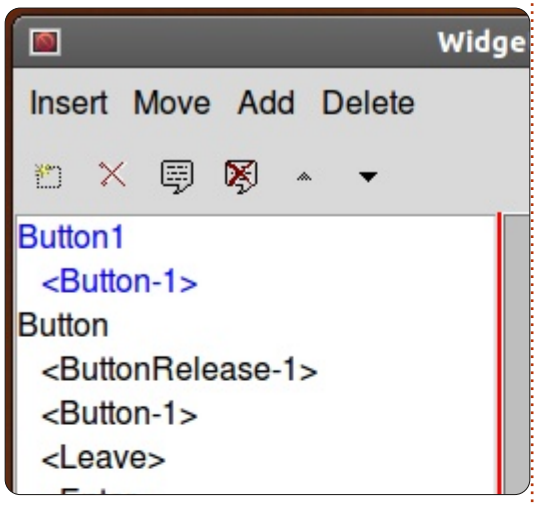
Dans la fenêtre de lancement, cherchez le menu Gen_Python et



cliquez dessus. Sélectionnez Generate Python et un nouveau formulaire apparaît.

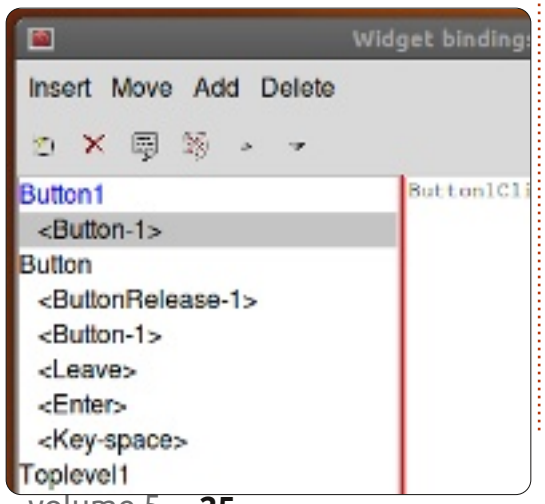
Page a généré (comme son nom l'indique) le code Python à notre place et l'a placé dans une fenêtre pour qu'on puisse le voir. Au bas de ce formulaire, il y a trois boutons : Save, Run et Close.

Cliquez sur Save. Si, à ce stade, vous deviez regarder dans votre dossier PageProjects, vous verriez le fichier python (TestForm.py). Maintenant, cliquez sur le bouton Run. En quelques secondes, vous verrez le projet démarrer. Le bouton n'est pas connecté à quoi que ce soit encore, il ne fera donc rien si vous cliquez dessus. Il suffit de fermer le formulaire avec le



« X » dans le coin de la fenêtre. Maintenant, fermez la fenêtre de console Python avec le bouton de fermeture en bas à droite.

De retour à notre formulaire principal, sélectionnez le bouton Exit et



faites un clic droit dessus. Sélectionnez « Bindings… ». Dans le menu se trouve un ensemble de boutons.

Le premier sur la gauche vous permet de créer une nouvelle liaison. Cliquez sur « Bouton-1 ». Cela nous permet de paramétrer la liaison pour le bouton gauche de la souris. Dans la fenêtre sur la droite, tapez « Button1Click ».

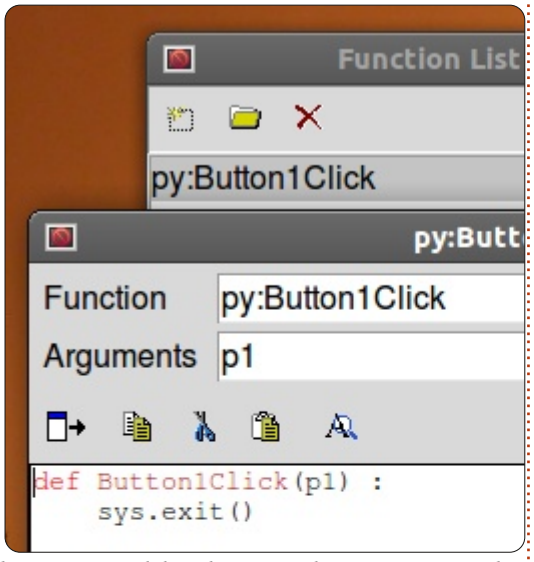
Enregistrez et générez le code python à nouveau. Faites défiler le code dans la console Python jusqu'à la fin du fichier. Au-dessus du code de la « class Test_Form » se trouve la fonction dont nous venons de demander la création. Notez qu'à ce stade, il est simplement transmis. Regardez plus loin vers le bas et vous

verrez le code qui crée et contrôle notre bouton. Tout est fait pour nous déjà. Toutefois, nous devons encore dire au bouton ce qu'il faut faire. Fermez la console Python et continuons.

Sur la fenêtre de lancement, cli-

quez sur Window puis sélectionnez Function List. Ici, nous allons écrire notre méthode pour fermer la fenêtre.

Le premier bouton à gauche est le



bouton Add. Cliquez dessus. Dans la zone Function, tapez « py:Button1Click » et, dans la zone Arguments, tapez « 1 » et modifiez le texte dans la zone inférieure à :

```
def Button1Click(p1):
        sys.exit()
```

Cliquez sur la coche et nous avons terminé avec cela.

Ensuite, nous devons lier cette routine au bouton. Sélectionnez le bouton dans le formulaire, faites un clic droit dessus et sélectionnez « Bindings… ». Comme précédemment, cliquez sur le bouton le plus à gauche sur la barre d'outils et sélectionnez le Button-1. C'est l'événement correspondant au clic gauche de la souris. Dans la boîte de texte à droite, entrez « Button1Click ». Assurez-vous d'utiliser la même casse que pour la fonction que nous venons de créer. Cliquez sur la coche sur le côté droit. Maintenant, sauvegardez et générez votre code python.

Vous devriez voir le code suivant vers le bas, mais en dehors de la classe Test_Form :

```
def Button1Click(p1) :

sys.exit()
```

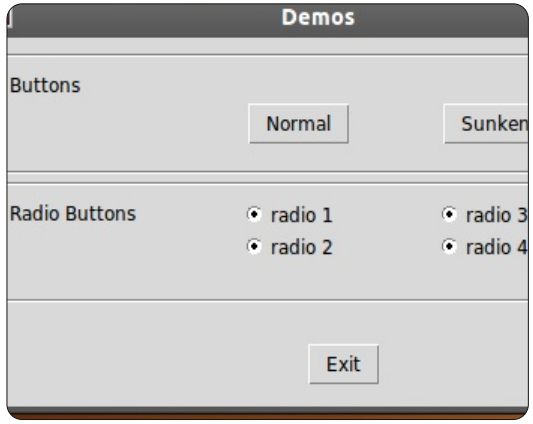
Et la dernière ligne de la classe devrait être :

```
self.Button1.bind('<Button-1>',Button1Click)
```

Maintenant, si vous exécutez votre code et cliquez sur le bouton Exit, le formulaire doit se fermer correctement.

Pour aller plus loin

Maintenant, nous allons faire quelque chose de plus compliqué. Nous allons créer une démo montrant quelques-uns des widgets qui sont disponibles. D'abord fermez Page et redémarrez-le. Ensuite, créez un nouveau formulaire Toplevel. Ajoutez deux cadres, l'un au-dessus de l'autre et étirez-les pour prendre à peu près toute la largeur du formulaire. Dans le cadre du haut, placez une étiquette de texte et, en utilisant l'éditeur attributs, modifiez le texte à « Buttons: ». Ensuite, ajoutez deux boutons dans le plan horizontal. Modifiez le texte de celui de gauche en « Normal » et celui de droite en « Sunken » [Ndt : en creux]. Alors que le bouton Sunken est



sélectionné, changez le relief de « Sunken » et nommez-le btnSunken. Nommez le bouton « Normal » « btnNormal ». Enregistrez ce projet avec le nom « Demos.tcl ».

Ensuite, placez dans le cadre inférieur une étiquette de texte « Radio Buttons » et quatre boutons radio comme dans l'image ci-dessous. Enfin, placez un bouton Exit en dessous du cadre inférieur.

Avant de travailler sur les liaisons, nous allons créer nos fonctions de clic. Ouvrez la liste de fonctions et créez deux fonctions. Le premier devrait être appelé btnNormalClicked et l'autre btnSunkenClicked. Assurez-vous d'inclure p1 dans la boîte d'arguments. Voici le code que vous devez avoir pour eux :

```
def btnNormalClicked(p1):

print "Normal Button Clicked"

def btnSunkenClicked(p1) :

print "Sunken Button Clicked"
```

Ajoutons nos liaisons aux boutons. Pour chaque bouton, faites un clic droit dessus, sélectionnez « Bindings… » et ajoutez, comme avant, une liaison aux fonctions que nous avons créées. Pour le bouton normal, cela sera « btnNormalClicked » et pour le bouton creux cela sera « btnSunkenClicked ». Enregistrez et générez votre code. Maintenant, si vous testiez le programme avec l'option « Run » de la console Python et cliquiez sur un des boutons, vous ne verriez rien se produire. Toutefois, lorsque vous fermerez l'application, vous devriez voir les réponses écrites. Ceci est normal pour Page et si vous l'aviez simplement exécuté à partir de la ligne de commande comme vous le feriez nor-

malement, les choses devraient fonctionner comme prévu.

Maintenant, passons à nos boutons radio. Nous les avons regroupés en deux groupes « clusters ». Les deux premiers (Radio 1 et Radio 2) formeront le groupe 1 et les deux autres seront le groupe 2. Cliquez sur Radio 1 et dans l'éditeur d'attributs, définissez la valeur à 0 et la variable à « rbc1 ». Définissez la variable pour Radio 2 à « rbc1 » et la valeur à 1. Faites la même chose pour Radio 3 et Radio 4, mais pour les deux réglez la variable à « rbc2 ». Si vous voulez, vous pouvez améliorer le clic des boutons radio et imprimer quelque chose dans le terminal, mais pour l'instant, la chose importante est que les groupes fonctionnent. Cliquer sur Radio1 désélectionnera Radio2 et n'aura pas d'influence sur Radio3 ou Radio4, et de même pour Radio2 et ainsi de suite.

Enfin, vous devez créer une fonction pour le bouton Exit et la lier au bouton, comme nous l'avons fait dans le premier exemple.

Si vous avez suivi depuis le début nos autres applications Tkinter, vous devriez être en mesure de comprendre le code montré ci-dessus à droite. Sinon, merci de retourner dans quelques numéros précédents pour lire une présentation complète de ce code.

```
def set_Tk_var():
# These are Tk variables passed to Tkinter and must
# be defined before the widgets using them are created.
global rbc1
rbc1 = StringVar()
global rbc2
rbc2 = StringVar()
def btnExitClicked(p1) :
sys.exit()
def btnNormalClicked(p1) :
print "Normal Button Clicked"
def btnSunkenClicked(p1) :
print "Sunken Button Clicked"
```

Vous pouvez voir qu'utiliser Page rend le processus de conception de base beaucoup plus facile que de le faire vous-même. Nous avons seulement commencé à examiner ce que peut faire Page et nous allons faire quelque chose de beaucoup plus réaliste la prochaine fois.

Le code python peut être trouvé sur pastebin à : http://pastebin.com/ts3MKyCZ.

Une note avant de terminer pour ce mois-ci. Vous avez sans doute remarqué que j'ai manqué quelques numéros. Cela est dû au fait que ma femme a été diagnostiquée avec un cancer l'an dernier. Même si j'ai vraiment essayé d'empêcher les choses de tomber à travers les mailles du filet, un certain nombre de choses y sont passées. Une de ces choses est mon ancien domaine/site web www.thedesignatedgeek.com. J'ai fait une grossière erreur et en ai raté le renouvellement. Pour cette raison, le domaine a été vendu sans mon consentement. J'ai mis en place www.thedesignatedgeek.net avec tous les vieux trucs. Je vais travailler dur le mois prochain pour mettre tout cela à jour.

Rendez-vous la prochaine fois.

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.com.

ubuntu uk podcast

Le Podcast Ubuntu couvre toutes les dernières nouvelles et les problèmes auxquels sont confrontés les utilisateurs de Linux Ubuntu et les fans du logiciel libre en général. La séance s'adresse aussi bien au nouvel utilisateur qu'au plus ancien codeur. Nos discussions portent sur le développement d'Ubuntu, mais ne sont pas trop techniques. Nous avons la chance d'avoir quelques supers invités, qui viennent nous parler directement des derniers développements passionnants sur lesquels ils travaillent, de telle façon que nous pouvons tous comprendre ! Nous parlons aussi de la communauté Ubuntu et de son actualité.

Le podcast est présenté par des membres de la communauté Ubuntu Linux du Royaume-Uni. Il est couvert par le Code de Conduite Ubuntu et est donc adapté à tous.

L'émission est diffusée en direct un mardi soir sur deux (heure anglaise) et est disponible au téléchargement le jour suivant.

podcast.ubuntu-uk.org

TUTORIEL

Débuter Python - Partie 31

Greg Walters

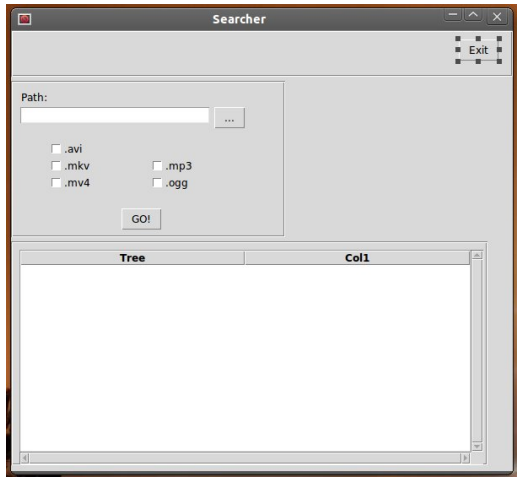
Après notre dernière rencontre, vous devriez avoir une assez bonne idée de la façon d'utiliser Page. Sinon, allez vite lire l'article du mois dernier. Nous allons continuer cette fois-ci en créant une application de liste de fichiers avec une interface graphique. Le but ici est de créer une application graphique qui va récursivement parcourir un répertoire, en cherchant des fichiers avec un ensemble défini d'extensions, et afficher le résultat dans une vue arborescente. Pour cet exemple, nous allons chercher les fichiers multimédias avec les extensions .avi, .mkv, .mv4, .mp3 et .ogg.

Cette fois, le texte peut sembler un peu laconique dans la partie conception. Tout ce que je vais faire, c'est vous donner des indications pour le placement des widgets, ainsi que les attributs et les valeurs requises, de cette façon :

```
widget

attribut: valeur
```

Je ne citerai les chaînes de texte que lorsque cela sera nécessaire. Par exemple, pour l'un des boutons, le texte doit être réglé sur « … ».

Voici à quoi va ressembler l'interface graphique de notre application :



Comme vous pouvez le voir, nous avons notre formulaire principal, un bouton pour quitter, une boîte de saisie de texte avec un bouton qui va appeler une boîte de dialogue pour demander le répertoire, cinq cases à cocher pour sélectionner les types d'extension, un bouton « ALLER ! » pour effectivement commencer le traitement et une arborescence pour afficher notre production.

Nous pouvons commencer. Lancez Page et créez un nouveau widget principal. En utilisant l'éditeur d'attributs, définissez les attributs suivants :

```
alias: rechercher
titre: rechercher
```

Assurez-vous de sauvegarder souvent. Lorsque vous enregistrez le fichier, donnez-lui le nom « Rechercher ». Rappelez-vous, Page ajoute l'extension .tcl à votre place et quand vous générerez le code python il sera sauvegardé dans le même dossier.

Ensuite, ajoutez un cadre. Il devrait se placer tout en haut du cadre principal. Définissez les attributs comme suit :

```
Largeur : 595
Hauteur : 55
Position x : 0
Position y : 0
```

Dans ce cadre, ajoutez un bouton. Ce sera notre bouton Quitter.

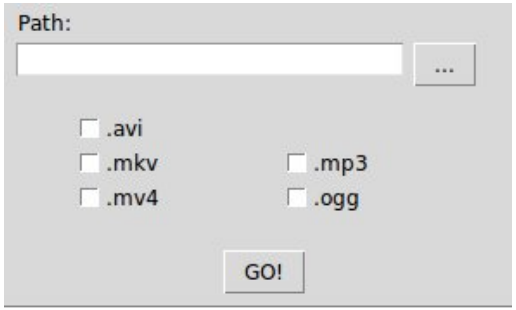
```
Alias : btnQuitter
Texte : Quitter
```

Déplacez-le au centre de la fenêtre, ou alors sur le côté droit. J'ai mis le mien à X = 530 et Y = 10.

Créez un autre cadre :

```
Largeur : 325
Hauteur : 185
Position y : 60
```

Voici à quoi ce cadre va ressembler, pour vous guider à travers cette section (colonne suivante).



Dans ce cadre, ajoutez une étiquette. Définissez l'attribut texte à « Chemin : ». Déplacez-le en haut à gauche de la fenêtre.

Dans le même cadre, ajoutez un widget de saisie :

```
Alias : txt Chemin
Texte : CheminFichier
Largeur : 266
Hauteur : 21
```

Ajoutez un bouton à droite de la zone de saisie :

```
Alias : btnCheminRecherche
Texte : « … » (sans guillemets).
```

Ajoutez cinq (5) cases à cocher. Mettez-les dans l'ordre suivant :

```
x
x   x
x   x
```

Les trois cases à cocher de gauche

sont pour les fichiers vidéo et les deux de droite sont pour les fichiers audio. Nous allons d'abord traiter les trois de gauche, puis les deux de droite.

```
Alias: chkAVI
Texte : ".avi" (sans guillemets)
Variable : VchkAVI

Alias : chkMKV
Texte : ".mkv" (sans guillemets)
Variable : VchkMKV

Alias: chkMV4
Texte : ".mv4"(sans guillemets)
Variable: VchkMV4

Alias : chkMP3
Texte : ".mp3" (sans guillemets)
Variable : VchkMP3

Alias : chkOGG
Texte : ".ogg" (sans guillemets)
Variable : VchkOGG
```

Enfin ajoutez dans ce cadre un bouton quelque part en dessous des cinq cases à cocher et un peu centré à l'intérieur du cadre :

```
Alias : btnAller
Texte : ALLER
```

Maintenant, ajoutez un autre cadre en dessous du précédent :

```
Largeur : 565
Hauteur : 265
```

J'ai placé le mien à environ X = 0 et Y = 250. Vous pourriez avoir à redimensionner votre formulaire principal pour voir l'affichage en entier. Dans ce cadre, ajoutez un widget Scrolledtreeview (vue arborescente avec ascenseur) :

```
Largeur : 550
Hauteur : 254
Position X : 10
Position Y : 10
```

Voilà. Nous avons conçu notre interface graphique. Maintenant tout ce qu'il reste à faire est de créer notre liste de fonctions et de lier ces fonctions à nos boutons.

Dans la fenêtre de liste des fonctions, cliquez sur le bouton Nouveau (le bouton le plus à gauche). Ceci nous amène à l'éditeur de nouvelle fonction. Modifiez le texte dans la zone de saisie Fonction en remplaçant « py:xxx » par « py:btnClicQuitter() ». Dans la zone de texte de l'argument saisissez « p1 ». Dans la zone de saisie multilignes du bas, changez le texte en :

```
def btnClicQuitter(p1):

    sys.exit()
```

Notez que ce n'est pas indenté. Page le fera pour nous quand il créera le fichier python.

Ensuite, créez une autre fonction appelée btnClicAller. N'oubliez pas d'ajouter un paramètre nommé « p1 ». Laissez l'instruction « pass » ; nous changerons cela plus tard.

Enfin, ajoutez une autre fonction appelée « btnCheminRecherche ». Encore une fois, laissez l'instruction « pass ».

En tout dernier lieu, nous devons relier les boutons et les fonctions que nous venons de créer.

Faites un clic droit sur le bouton Quitter que nous avons créé, sélectionnez Lier. Une grande boîte apparaîtra. Cliquez sur le bouton Nouvelle liaison, cliquez sur bouton-1 et remplacez le mot « A FAIRE » dans la boîte de saisie de texte de droite par « btnClicQuitter ». NE METTEZ PAS les parenthèses () ici.

Liez le bouton ALLER à la fonction btnClicAller et le bouton « … » à btnClicCheminRecherche.

Sauvegardez votre interface graphique et générez le code python.

Maintenant tout ce qu'il reste à faire est de créer le code qui « agglutine » l'interface graphique.

Ouvrez le code que nous venons de générer dans votre éditeur de texte favori. Commençons par examiner ce que Page a créé pour nous.

Au début du fichier se trouve l'en-tête standard python et une déclaration d'importation unique pour importer la bibliothèque système (sys). Ensuite vient du code plutôt confus (à première vue). C'est simplement pour examiner la version de python avec laquelle vous essayez d'exécuter l'application, puis pour importer les versions correctes des bibliothèques Tkinter. À moins que vous n'utilisiez Python 3.x, vous pouvez tout simplement ignorer les deux derniers.

Nous allons modifier la portion de code 2.x dans quelques instants pour importer d'autres modules Tkinter.

Arrive ensuite la routine « vp_start_gui() ». C'est la routine principale du programme. Ceci met en place notre interface, définit les variables dont nous avons besoin et appelle ensuite la boucle principale Tkinter. Vous remarquerez peut-être la ligne « w = None » juste en dessous. Elle n'est pas indentée et n'a pas besoin de l'être.

Ensuite viennent deux routines (create_Rechercher et destroy_Rechercher) qui sont utilisées pour remplacer la routine principale si nous utilisons cette application comme une bibliothèque. Nous n'avons pas besoin de nous inquiéter à ce sujet.

Arrive ensuite la routine « initialise_var_Tk ». Nous définissons les variables Tkinter utilisées qui doivent être mises en place avant de créer les widgets. Vous pouvez sans doute reconnaître la variable texte pour le widget de saisie CheminFichier et les variables de nos cases à cocher. Les trois routines suivantes sont les fonctions que nous avons créées en utilisant l'éditeur de fonctions et une fonction « init() ».

Exécutez le programme maintenant. Notez que les cases à cocher contiennent des coches grisées. Nous ne voulons pas cela dans notre application finale, nous allons donc créer un peu de code pour les faire disparaître avant que le formulaire ne soit affiché à l'utilisateur. La seule chose qui fonctionne à part les cases à cocher est le bouton Quitter.

Utilisez-le pour terminer le programme.

Maintenant, nous allons jeter un coup d'oeil à la classe qui contient effectivement la définition de l'interface graphique. Il s'agit de la classe « Chercheur ». C'est là que tous les widgets sont définis et placés dans notre formulaire. Vous devez être familier avec cela maintenant.

Deux classes de plus sont créées, elles contiennent le code pour gérer l'arborescence qui défile. Nous n'avons pas à changer tout cela. Tout a été créé pour nous par Page.

Revenons maintenant au début du code et commençons à le modifier.

Nous avons besoin d'importer quelques modules de bibliothèque de plus ; pour cela ajoutez en dessous de la déclaration « import sys » :

```
import os

from os.path import join,
getsize, exists
```

Maintenant, trouvez la section qui contient la ligne « py2 = True ». Comme nous l'avons dit, c'est la section qui traite des importations tkinter pour Python version 2.x. En dessous de « import ttk », nous avons besoin d'ajouter ce qui suit pour utiliser la bibliothèque FileDialog. Nous avons également besoin d'importer le module tkFont :

```
import tkFileDialog

import tkFont
```

Ensuite nous devons ajouter quelques variables à la routine « initialise_var_Tk() ». En bas de la routine, ajoutez les lignes suivantes :

```
global exts, FileList

exts = []

ListeFichiers=[]
```

Ici, nous créons deux variables globales (exts et ListeFichiers) qui seront utilisées plus tard dans notre code. Les deux sont des listes. « exts » est une liste des extensions que l'utilisateur sélectionne dans l'interface. « ListeFichiers » contient une liste des fichiers correspondants à la recherche effectuée par l'utilisateur. Nous allons l'utiliser pour remplir le widget de vue arborescente.

Puisque notre « btnClicQuitter » est déjà créé pour nous par Page, nous allons nous occuper de la routine « btnClicAller ». Commentez la déclaration pass et ajoutez le code de sorte qu'il ressemble à ceci :

```
def btnClicAller(p1) :

#pass

ConstruireExts()

chemin = CheminFichier.get()

e1 = tuple(exts)

Parcourir(chemin,e1)

ChargerDonnees()
```

C'est la routine qui est appelée lorsque l'utilisateur clique sur le bouton « ALLER ». Nous appelons une routine nommée « ConstruireExts » qui crée la liste des extensions que l'utilisateur a sélectionnée. Puis nous récupérons le chemin que l'utilisateur a choisi dans la boîte de dialogue de demande de répertoire et l'assignons à la variable chemin. Nous créons ensuite un tuple à partir de la liste des extensions, ce qui est nécessaire quand nous vérifions les fichiers. Nous appelons ensuite une routine appelée « Parcourir » en lui passant le répertoire cible et le tuple des extensions.

Enfin, nous appelons une routine nommée « ChargerDonnees ».

Ensuite, nous devons étoffer la routine « btnClicCheminRecherche ». Commentez la déclaration pass et modifiez le code pour qu'il ressemble à ceci :

```
def
btnClicCheminRecherche(p1) :

     #pass

     chemin =
tkFileDialog.askdirectory()
#**self.file_opt)

     CheminFichier.set(chemin)
```

Puis vient la routine init. À nouveau, le code doit ressembler à ceci :

```
def init():

    #pass

    # se lance apres la creation
des fenetres et des widgets...

    global VueArborescente

    InitialiserCases()

    VueArborescente =
w.Scrolledtreeview1


InitialiserVueArborescente()
```

Ici, nous créons une variable globale appelée VueArborescente. Nous appelons ensuite une routine qui efface les contrôles gris dans les cases à cocher, affectons la variable VueArborescente pour pointer vers l'arborescence avec ascenseurs de notre formulaire et appelons

InitialiserVueArborescente pour définir les en-têtes pour les colonnes. Voici le code de la routine InitialiserCases qui doit être la suivante :

```
def InitialiserCases():

    VchkAVI.set('0')

    VchkMKV.set('0')

    VchkMP3.set('0')

    VchkMV4.set('0')

    VchkOGG.set('0')
```

Ici, tout ce que nous faisons, c'est de définir les variables (ce qui définit automatiquement l'état d'activation dans nos cases à cocher) à 0. Si vous vous souvenez, à chaque fois qu'on clique sur la case, cette variable est automatiquement mise à jour. Si la variable est modifiée par notre code, la case à cocher répond également. Maintenant (en haut à droite) nous allons nous occuper de la routine qui établit la liste des extensions à partir de ce que l'utilisateur a cliqué.

Essayez de vous rappeler mon neuvième article dans le FCM nº 35. Nous avons écrit du code pour créer un catalogue de fichiers MP3. Nous allons utiliser une version abrégée de cette routine (au milieu à droite). Reportez-vous au FCM nº 35 si vous avez des questions au sujet de cette routine.

Ensuite (en bas à droite) nous appelons la routine InitialiserVueArborescente. C'est assez simple. Nous définissons une variable « TitresColonnes » avec les rubriques que nous voulons dans chaque colonne de l'arborescence. Nous utilisons une liste pour cela. Nous réglons ensuite l'attribut titre de chaque colonne. Nous réglons également la largeur de colonne à la taille de cet en-tête.

Enfin, nous devons créer la routine « ChargerDonnees » (page suivante, en haut à droite) qui est l'endroit où nous chargeons nos données dans l'arborescence. Chaque ligne de l'arborescence est une entrée dans la variable de type liste ListeFichiers. Nous devons également ajuster la largeur de chaque colonne (à nouveau) pour correspondre à la taille des données de la colonne.

C'est tout pour la première vue de l'application. Exécutez-la et regardez ce que ça fait. Notez que si vous avez un grand nombre de fichiers à parcourir, vous aurez l'impression que le programme ne répond pas. C'est quelque chose qui doit être corrigé. Nous allons créer des routines pour modifier notre curseur de la valeur par défaut à un curseur en forme de montre et vice-versa pour que l'utilisateur soit au courant quand nous faisons quelque chose qui prend beaucoup de temps.

Dans la routine « initialise_var_Tk », ajoutez le code suivant à la fin :

```
global
CurseurOccupe,PreCurseurOccupe,
WidgetsOccupes

CurseurOccupe = 'watch'

PreCurseurOccupe = None

WidgetsOccupes = (racine, )
```

Ici, nous mettons en place des variables globales, nous les initialisons, puis nous réglons le(s) widget(s) (dans WidgetsOccupes) pour lesquels nous souhaitons gérer le changement de curseur. Dans ce cas, nous avons mis racine qui est notre fenêtre entière. Remarquez que c'est un tuple.

```
def ConstruireExts():
    if VchkAVI.get() == '1':
        exts.append(".avi")
    if VchkMKV.get() == '1':
        exts.append(".mkv")
    if VchkMP3.get() == '1':
        exts.append(".mp3")
    if VchkMV4.get() == '1':
        exts.append(".mv4")
    if VchkOGG.get() == '1':
        exts.append(".ogg")
```

Ensuite, nous créons deux routines pour

```
def Parcourir(chemin,extensions):
    rcntr = 0
    liste = []
    for racine, reps, fics in os.walk(chemin):
        rcntr += 1  # nombre de repertoires parcourus
        for fic in [f for f in fics if f.endswith(extensions)]:
            liste.append(fic)
            liste.append(racine)
            ListeFichiers.append(liste)
            liste=[]
```

```
def InitialiserVueArborescente():
    global TitresColonnes
    TitresColonnes = ['Nom fichier','Chemin']
    VueArborescente.configure(columns=TitresColonnes,
        show="headings")
    for col in TitresColonnes:
        VueArborescente.heading(col, text = col.title(),
                command = lambda c = col: sortby(VueArborescente, c, 0))
        ## ajuste la largeur de colonne au titre
        VueArborescente.column(col, width =
tkFont.Font().measure(col.title()))
```

modifier et remettre le curseur. D'abord la routine qui modifie, que nous appelons « DebutOccupation ». Insérez le code que vous voyez au milieu à droite après la routine « ChargerDonnees ».

Nous vérifions d'abord si une valeur a été passée à « nouveaucurseur ». Sinon, nous mettons par défaut CurseurOccupe. Puis nous parcourons le tuple WidgetsOccupes et réglons le curseur sur ce que nous voulons.

Maintenant, mettez le code que vous voyez tout à fait en bas.

Dans cette routine, nous réinitialisons simplement le curseur pour les widgets dans notre tuple WidgetsOccupes à notre curseur par défaut.

Enregistrez et exécutez votre programme. Vous devrez voir que le curseur change chaque fois que vous avez une longue liste de fichiers à parcourir.

Cette application ne fait vraiment pas grand chose, mais elle vous a montré comment utiliser Page pour développer très rapidement. Avec l'article d'aujourd'hui, vous pouvez voir qu'une bonne conception de votre interface graphique à l'avance peut rendre le processus de développement facile et relativement indolore.

Le fichier tcl est enregistré sur pastebin : http://pastebin.com/AA1kE4Dy (en anglais) et le code Python est enregistré ici : http://pastebin.com/WYK2SKQj.

À la prochaine fois !

```
def ChargerDonnees():
    global TitresColonnes
    for c in ListeFichiers:
        VueArborescente.insert('','end',values=c)
        # ajuste la largeur de colonne si necessaire pour chaque valeur
        for ix, val in enumerate(c):
            larg_col = tkFont.Font().measure(val)
            if VueArborescente.column(TitresColonnes[ix],width=None)<larg_col:
                VueArborescente.column(TitresColonnes[ix], width=larg_col)
```

```
def debutOccupation(nouveaucurseur=None):
    global PreCurseurOccupe
    if not nouveaucurseur:
        nouveaucurseur = CurseurOccupe
    nouveauPreCurseursOccupes = {}
    for composant in WidgetsOccupes:
        nouveauPreCurseursOccupes[composant] = composant['cursor']
        composant.configure(cursor=nouveaucurseur)
        composant.update_idletasks()
        PreCurseurOccupe = (nouveauPreCurseursOccupes, PreCurseurOccupe)
```

```
def finOccupation():
    global PreCurseurOccupe
    if not PreCurseurOccupe:
        return
    ancienPreCurseursOccupes = PreCurseurOccupe[0]
    PreCurseurOccupe = PreCurseurOccupe[1]
    for composant in WidgetsOccupes:
        try:
            composant.configure(cursor=ancienPreCurseursOccupes[composant])
        except KeyError:
            pass
        composant.update_idletasks()
```

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume six

Parties 32 à 38

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

En réponse aux requêtes des lecteurs, nous avons réuni le contenu de certains articles consacrés à la programmation en python.

Pour l'instant, il s'agit d'une réédition directe de la série Débuter Python, parties 32 à 38, des numéros 60 à 67 ; en effet, nous avons permis à l'incomparable professeur en Python Greg Walters de sauter le numéro 66 parce qu'il a été très sage.

Veuillez considérer l'origine de la publication ; les versions actuelles du matériel et des logiciels peuvent différer de ceux que nous présentons, ainsi vérifiez bien votre matériel et la version de vos logiciels avant d'émuler les tutoriels de cette édition spéciale. Vous pouvez installer des versions de logiciels plus récentes ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

SiteWeb :
http://www.fullcirclemagazine.org/

Forums :
http://ubuntuforums.org/forumdisplay.php?f=270

IRC : #fullcirclemagazine on chat.freenode.net

Équipe éditoriale :
Rédacteur en chef : Ronnie Tucker (pseudo : RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia (pseudo : admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcast : Robin Catling (pseudo : RobinCatling)
podcast@fullcirclemagazine.org
Dir. comm. : Robert Clipsham (pseudo : mrmonday) -
mrmonday@fullcirclemagazine.org

TUTORIEL

Débuter Python - Partie 32

Greg Walters

Je dois l'avouer, j'aime ma tablette Android. Alors que je l'utilise tous les jours, elle n'est pas encore un remplaçant pour mon ordinateur de bureau. Et je dois aussi admettre que ce pourquoi je l'utilise principalement est à peu près ce pourquoi tout le monde utilise la sienne : naviguer sur le Web, écouter de la musique, regarder des vidéos, jouer à des jeux et ainsi de suite. J'essaie de la justifier en ayant des applications qui ont un rapport avec des listes de courses et de tâches à faire, la recherche d'essence pas chère, des choses amusantes pour mon petit-fils, etc. C'est vraiment un jouet pour moi en ce moment. Pourquoi utiliser une tablette tactile amusante pour faire votre liste de courses ? Avouons-le… Ce sont les regards d'envie que les gens me jettent dans le magasin quand ils me voient pousser le caddy dans l'allée et tapoter ma tablette pour barrer les éléments de la liste. Ah, le facteur geek RÈGNE ! Bien sûr, je peux utiliser le dos d'une vieille enveloppe pour faire ma liste. Mais ce ne serait ni cool ni geeky, hein, n'est-ce pas ?

Comme 99 % des hommes geeks mariés dans le monde, je suis marié à une femme non-geek. Une merveilleuse femme aimante, bien sûr, mais une non-geek qui, lorsque je commence à baver sur le dernier gadget, soupire et dit quelque chose comme : « Eh bien, si tu penses VRAIMENT que nous avons besoin de ça… ». Puis elle m'envoie le même regard que je lui envoie quand elle caresse affectueusement la 50e paire de chaussures du magasin.

En toute honnêteté, il n'a pas été difficile de ramener la première tablette à la maison. Je l'ai achetée pour ma femme alors qu'elle suivait un traitement de chimiothérapie. Elle a essayé d'utiliser un ordinateur portable pendant quelque temps, mais la chaleur et le poids sur ses genoux étaient insupportables au bout d'un certain temps. Les e-books sur un ordinateur portable n'étaient pas sa tasse de thé, alors quand elle a essayé de lire, elle a dû jongler entre le livre, l'ordinateur portable et le lecteur mp3, tout en étant attachée à un transat avec des tubes dans le bras la remplissant de vilains produits chimiques. Lorsque je lui ai offert la tablette, ce fut le meilleur des mondes. Elle pouvait lire un livre électronique, écouter de la musique, regarder la télé, naviguer sur le Web, vérifier ses mails, mettre à jour son blog sur le cancer, suivre ses amis sur facebook et jouer à des jeux, le tout sur un appareil qui était léger et sympa. Si elle était fatiguée, elle pouvait tout simplement la glisser sur le côté entre elle et le transat (ou le lit quand elle était à la maison pour essayer de reprendre des forces). Bien mieux qu'un ordinateur portable encombrant et un livre, un lecteur mp3, une télécommande et plus encore.

Alors qu'elle se faisait remplir de produits chimiques nocifs, je réquisitionnais une table et une chaise dans le coin de la salle de traitement, à proximité d'une prise de courant, et j'essayais de travailler sur mon vieux portable de six ans. Entre les projets, je faisais des recherches sur la programmation Android. J'ai découvert que la plupart des programmes pour Android se faisait en Java. J'étais presque résigné à ré-apprendre le Java lorsque je suis tombé sur quelques outils qui permettent de programmer en Python pour le système d'exploitation Android. Un de ces outils est appelé « SL4A ». SL4A veut dire Scripting Layer for Android (Couche de Script pour Android). C'est ce sur quoi nous allons nous concentrer dans les deux prochains articles. Dans celui-ci, en fait, nous allons nous concentrer sur la mise en place de SL4A sous Android.

De nombreuses pages web montrent comment charger SL4A dans l'émulateur Android pour ordinateur de bureau. Nous allons essayer de faire cela une autre fois, mais pour l'instant nous allons jouer avec l'appareil Android lui-même. Pour installer SL4A sur votre appareil Android, allez à http://code.google.com/p/android-scripting/ ; vous y trouverez le fichier d'installation pour SL4A. Ne soyez pas totalement perdu ici. Il y a un code QR sur

lequel vous appuyer pour télécharger l'APK. Assurez-vous d'avoir activé l'option « Unknown Sources (Sources inconnues) » dans les paramètres Application. C'est un téléchargement rapide. Une fois que vous l'avez téléchargé et installé, trouvez l'icône et appuyez dessus. Ce que vous verrez est un écran noir, plutôt décevant, disant « Scripts…No matches found » (Scripts… Aucune correspondance trouvée). Ce n'est pas grave. Cliquez sur le bouton menu et sélectionnez View (Afficher). Vous verrez un menu. Sélectionnez Interpreters (interprètes). Ensuite, sélectionnez de nouveau le menu, puis sélectionnez Add (Ajouter). Dans le menu suivant, sélectionnez Python 2.6.2. Cela devrait vous demander de lancer un navigateur pour télécharger Python pour Android. Une fois qu'il est installé, sélectionnez Open (Ouvrir). Vous obtiendrez un menu à l'écran avec les choix Install, Import Modules, Browse Modules (parcourir les modules), et Uninstall Modules. Sélectionnez l'option Install. Maintenant Python va se télécharger et s'installer avec d'autres modules supplémentaires. De plus, vous aurez des exemples de scripts. Enfin, appuyez sur le bouton de retour et vous verrez Python 2.6.2 installé dans l'écran des interprètes. Appuyez de nouveau sur le bouton de retour et vous verrez une liste de quelques exemples de scripts python.

> **Ce que vous verrez est un écran noir, plutôt décevant. Ce n'est pas grave.**

C'est tout ce que nous allons faire cette fois-ci. Tout ce que je voulais faire, c'était vous mettre en appétit. Explorez Python sous Android. Vous pouvez également visiter http://developer.android.com/sdk/index.html pour obtenir le SDK Android (Soft-ware Development Kit) pour votre bureau. Il comprend un émulateur Android afin que vous puissiez jouer avec tout de suite. La mise en place du SDK est vraiment très facile sous Linux et vous ne devriez pas avoir trop de mal.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Comment inclure des accents à partir du clavier

par Barry Smith

Si votre système Linux est en français, allemand ou espagnol et, par conséquent, exige des accents, ou si, de temps en temps, vous avez besoin d'utiliser des accents qui ne figurent pas dans les mots anglais, de nombreux utilisateurs ne savent pas qu'il existe un moyen très facile pour le faire à partir du clavier. Ce qui suit ne s'applique qu'à un clavier du Royaume-Uni.

Accent aigu
Appuyez sur Alt Gr + ; (point-virgule). Levez la main puis appuyez sur la voyelle souhaitée é.

Accent circonflexe
Appuyez sur Alt Gr + ' (apostrophe). Levez la main puis appuyez sur la voyelle souhaitée î.

Accent grave
Appuyez sur Alt Gr + # (dièse). Levez la main puis appuyez sur la voyelle souhaitée è.

Trema
Appuyez sur Alt Gr + [. Levez la main puis appuyez sur u ü.

ñ - Appuyez sur Alt Gr +]. Levez la main puis appuyez sur n ñ.

œ - Appuyez sur Maj + Alt Gr. Levez la main puis appuyez sur o puis appuyez sur e œ. Le œ n'apparaîtra pas avant que le e ne soit tapé.

Pour obtenir ¿ et ¡ (point d'exclamation et d'interrogation inversés) dont je me sers tout le temps en espagnol avant les questions et les exclamations, appuyez sur Alt Gr + Maj, en gardant les deux touches enfoncées, puis appuyez sur _ (souligné) pour ¿ ou tapez ! (point d'exclamation) pour ¡.

Si vous voulez une de ces lettres en capitales, il suffit d'appuyer sur Maj avant de taper de la lettre.

Débuter Python - Partie 33

Greg Walters

Ce mois-ci, nous allons mettre en place le SDK Android sur notre bureau Linux. Nous allons aussi créer un périphérique Android virtuel, installer SL4A et Python dessus, et faire un test rapide.

S'il vous plaît faites attention, ce n'est pas quelque chose qu'il faut faire sur une machine qui a moins de 1 Go de RAM. L'émulateur consomme une énorme quantité de mémoire. Je l'ai essayé sur un ordinateur portable fonctionnant sous Ubuntu avec seulement 512 Mo de RAM. Il fonctionne, mais il est VRAIMENT lent.

Voici une liste rapide de ce que nous allons faire. Nous allons y aller étape par étape dans une minute :

- Installer le JDK6 Java.
- Installer le pack de démarrage SDK Android.
- Créer et configurer les AVD.
- Tester AVD et installer SL4A et Python.

En réalité, nous devrions également installer Eclipse et le plugin Android ADT pour Eclipse, mais, puisque nous n'utiliserons pas Eclipse dans cette série d'articles, nous pouvons éviter cela. Si vous souhaitez les inclure, allez voir sur http://developer.android.com/sdk/installing.html pour voir toutes les étapes dans l'ordre suggéré. Nous allons maintenant commencer.

ÉTAPE 1 - Java JDK 6

D'après tout ce que j'ai lu et essayé, il faut vraiment utiliser la version de Sun. OpenJDK n'est pas censé fonctionner. Vous pouvez trouver des informations à ce sujet sur le web, mais voici les étapes que j'ai suivies. Dans un terminal, tapez les commandes suivantes :

```
sudo add-apt-repository
ppa:ferramroberto/java

sudo apt-get update

sudo apt-get install sun-
java6-jdk
```

Une fois que tout ceci est fait, vous devrez modifier votre fichier .bashrc pour régler « JAVA_HOME » pour que tout fonctionne correctement. J'ai utilisé gedit pour ajouter la ligne suivante à la fin du fichier :

```
export
JAVA_HOME="/usr/lib/jvm/java-
6-sun-1.6.0.06"
```

Enregistrez le fichier et passez à l'étape 2.

ÉTAPE 2 - Pack de démarrage Android SDK

Maintenant, la partie « marrante » commence. Rendez-vous sur http://developer.android.com/sdk/index.html. C'est là que se trouve le SDK. Téléchargez la dernière version pour Linux qui, au moment d'écrire ces lignes, est android-sdk_r18-linux.tgz. À l'aide du Gestionnaire d'archives, décompressez-la dans un dossier approprié. Je l'ai mise dans mon répertoire personnel. Tout fonctionne directement à partir de ce dossier, vous n'avez donc vraiment pas besoin d'installer quoi que ce soit. Ainsi, le chemin pour moi est /home/greg/android-sdk-linux. Allez dans ce dossier, puis allez dans le dossier des outils (« tools »). Vous y trouverez un fichier nommé « android ». C'est lui qui lance réellement le SDK. J'ai créé un lanceur sur mon bureau pour en faciliter l'accès.

À présent, la partie ennuyeuse. Exécutez le fichier android ; le gestionnaire de SDK Android va démarrer. Il va mettre à jour les plateformes qui sont disponibles. Je vous préviens maintenant que ce processus prendra un certain temps, alors ne vous embêtez pas si vous n'avez pas beaucoup de temps pour y faire face. Par souci de concision, je vous conseille de n'avoir qu'une plateforme pour commencer. Je vous suggère de commencer par Android 2.1, puisqu'en général si vous développez pour une ancienne plateforme, il ne devrait y avoir aucun problème d'exécution sur une nouvelle plateforme. Vous devez également récupérer l'ensemble des outils. Il suffit de cocher la case à côté de ces deux éléments, puis de cliquer sur le bouton d'installation. Une fois que vous avez obtenu la plateforme de votre choix et l'ensemble d'outils, vous êtes presque prêt à créer votre première machine virtuelle.

ÉTAPE 3 - Créer et configurer votre première AVD

Retournez dans le Gestionnaire de SDK Android, sélectionnez Outils (« Tools ») dans le menu principal, puis sélectionnez « Gérer les AVD ». Cela va ouvrir une nouvelle fenêtre. Puisque c'est la première fois, il n'y aura pas encore de périphérique virtuel configuré. Cliquez sur le bouton « Nouveau ». Cela ouvre une autre fenêtre où nous définissons les propriétés du périphérique virtuel Android. Voici les étapes que vous devrez suivre

pour mettre en place un dispositif émulateur Android simple :

• Définissez le nom de l'appareil. Ceci est important si vous avez plus d'un appareil.
• Réglez le niveau de plateforme cible.
• Définissez la taille de la carte SD (voir ci-dessous).
• Réglez la résolution.
• Créez le périphérique.

Par exemple, dans la zone de texte Nom, tapez « Test1 ». Pour la cible, sélectionnez Android 2.1 - API de niveau 7. Dans la boîte pour « Carte SD : », entrez 512 et assurez-vous que la liste affiche « Mio ». Dans « Skin », réglez la résolution à 800×600. (Vous pouvez jouer avec les autres paramètres de tailles.) Enfin, cliquez sur le bouton « Créer AVD ». Vous verrez alors un message disant que l'AVD a été créée.

ÉTAPE 4 - Test de l'AVD et installation de SL4A et Python

Maintenant, enfin, nous pouvons nous amuser un peu. Mettez en surbrillance l'AVD que vous venez de créer et cliquez sur le bouton Démarrer. Dans la boîte de dialogue qui apparaît, cliquez simplement sur le bouton « Lancer ». Vous devez alors attendre quelques minutes pour que le périphérique virtuel soit créé dans la mémoire et que la plateforme Android soit chargée et démarrée. (Nous reparlerons de l'accélération de ce processus dans un prochain article.)

Une fois que l'AVD a démarré et que vous avez l'écran d'accueil, vous allez installer SL4A. En utilisant le navigateur ou la boîte de recherche Google Web sur l'écran d'accueil, recherchez « sl4a ». Allez à la page des téléchargements et vous finirez par trouver la page web pour les téléchargements http://code.google.com/p/android-scripting/downloads/list.

Faites défiler la page jusqu'à ce que vous obteniez le lien sl4a_r5. Ouvrez le lien et tapez sur le lien « sl4a_r5.apk ». Remarquez que j'ai dit « tapez » plutôt que « cliquez ». Commencez à penser à votre doigt qui appuie sur l'écran plutôt que de cliquer avec la souris. Cela facilitera votre transition vers la programmation. Vous verrez le début de téléchargement. Vous pourriez avoir à tirer vers le bas la barre de notification en haut pour obtenir le fichier téléchargé. Tapez sur le fichier, puis sur le bouton d'installation.

Une fois le fichier téléchargé, vous verrez la possibilité d'ouvrir l'application téléchargée ou de taper sur « Terminé » pour quitter le programme d'installation. Ici, il faut taper sur « Ouvrir ».

Maintenant SL4A va démarrer. Vous verrez probablement une boîte de dialogue vous demandant si vous acceptez un suivi de l'utilisation. C'est à vous de décider si vous voulez accepter ou refuser. Avant d'aller plus loin, vous devriez connaître quelques raccourcis clavier qui vous aideront à vous déplacer. Comme nous n'avons pas un « vrai » appareil Android, les boutons Retour, Accueil et Menu ne sont pas disponibles. Vous en aurez besoin pour naviguer. Voici quelques raccourcis importants :

Retour - Échap
Accueil - Début
Menu - F2

Maintenant, nous voulons télécharger et installer Python dans SL4A. Pour faire cela, appuyez d'abord sur Menu (F2). Sélectionnez « Affichage » dans le menu. Maintenant, sélectionnez « Interprètes ». On dirait que rien ne se passe, mais appuyez sur Menu à nouveau (F2), puis sélectionnez « Ajouter » dans le menu contextuel. Maintenant, faites défiler vers le bas et sélectionnez « Python 2.6.2 ». Ceci va télécharger le paquet de base Python pour Android. Installez le paquet puis ouvrez-le. Vous verrez quatre options. Installer, importer des modules, parcourir les modules et désinstaller un module. Tapez sur Installer. Cela va démarrer le téléchargement et l'installation de tous les morceaux de la dernière version de Python pour Android. Cela peut prendre quelques minutes.

Une fois que tout est terminé, appuyez sur Retour (touche Échap) jusqu'à ce que vous arriviez à l'écran des interprètes SL4A. Maintenant tout est chargé pour que nous puissions jouer en Python sur Android. Tapez sur Python 2.6.2 et vous vous trouverez dans la ligne de commande standard de Python. C'est exactement comme la ligne de commande sur votre bureau. Saisissez les trois lignes suivantes, une à la fois, dans la ligne de commande. Assurez-vous d'attendre l'invite « > » à chaque fois.

```
import android

droid = android.Android()

droid.makeToast("Bonjour
depuis Python pour Android")
```

Après avoir tapé la dernière ligne et appuyé sur Entrée, vous verrez une fenêtre aux coins arrondis centrée en bas de la ligne de commande, qui dit : « Bonjour depuis Python pour Android ». C'est ce que fait la commande droid.makeToast.

Vous avez écrit votre premier script Python pour Android. Chouette, hein ?

Maintenant, nous allons créer un raccourci sur l'écran d'accueil d'Android. Tapez sur la touche Accueil (bouton Début). Si vous avez choisi la plateforme 2.1, vous devriez voir une barre de défilement à l'extrême droite de l'écran. Si vous avez choisi une autre plateforme, il se pourrait que ce soit un carré ou un rectangle composé de petits carrés. De toutes les façons, cela vous amène à l'écran des Applis. Tapez dessus et trouvez l'icône SL4A. Maintenant effectuez un « taper long » (clic long), qui créera un raccourci sur l'écran d'accueil. Déplacez le raccourci où vous le souhaitez.

Ensuite nous allons créer notre premier script sauvegardé. Retournez dans SL4A. Vous devriez voir les exemples de scripts fournis avec Python pour Android. Tapez sur le bouton Menu et sélectionnez « Ajouter ». Sélectionnez « Python 2.6.2 » dans la liste. Vous verrez l'éditeur de script. Au sommet se trouve la boîte de nom de fichier avec « .py » déjà rempli. En dessous se trouve la fenêtre de l'éditeur qui contient déjà les deux premières lignes de notre programme saisies pour nous. (Je les ai inclues ci-dessous en italique pour que vous puissiez le vérifier. Nous avons également utilisé ces deux lignes dans notre premier exemple.)

```
import android

droid = android.Android()
```

Maintenant saisissez les deux lignes suivantes dans le script python :

```
uname =
droid.dialogGetInput("Quel
est votre nom ?")

droid.makeToast("Bonjour
%s depuis Python pour
Android") % uname.result
```

La première ligne nouvelle crée une boîte de dialogue (droid.dialogGetInput()) qui demande son nom à l'utilisateur. La réponse est retournée à notre programme dans uname.result. Nous avons déjà utilisé la fonction droid.makeToast().

Nommez le fichier andtest1.py, puis tapez sur Terminé puis sur « Enregistrer et Exécuter ». Si tout s'est bien passé, vous devriez voir une boîte de dialogue vous demandant votre nom. Après l'avoir saisi, vous devriez voir l'alerte en bas de l'écran qui affiche : « Bonjour Votre nom depuis Python pour Android ».

C'est tout pour cette fois-ci. Pour l'instant, il y a une tonne de documentation gratuite sur SL4A sur le web. Vous pouvez jouer un peu tout seul jusqu'à la prochaine fois. Je vous suggère de commencer par http://code.google.com/p/android-scripting/wiki/Tutorials.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Débuter Python - Partie 34

Greg Walters

Cette fois-ci, nous allons en terminer avec l'utilisation de SL4A. Nous allons faire un programme plus important, puis l'envoyer à la machine virtuelle via ABD.

Occupons-nous d'abord de notre code. Pour cela, nous allons simplement essayer quelques-uns des « widgets » qui sont à notre disposition lorsqu'on utilise SL4A. Démarrez sur votre bureau à l'aide de votre éditeur de texte favori.

```
import android

droid = android.Android()
uname = droid.dialogGetInput("Hello","What's your name?")
droid.makeToast("Hello %s from python on Android!" % uname.result)
```

```
droid.dialogCreateAlert(uname.result,"Would you like to play a game?")
droid.dialogSetPositiveButtonText('Yes')
droid.dialogSetNegativeButtonText('No')
droid.dialogShow()
while True: #wait for events for up to 10 seconds...
      response = droid.eventWait(10000).result
      if response == None:
      break
      if response["name"] == "dialog":
      break
droid.dialogDismiss()
```

Entrez le code que vous voyez en haut à droite et sauvegardez-le (mais n'essayez pas de l'exécuter) en tant que « atest.py ».

La première ligne importe la bibliothèque android. Nous en créons une instance dans la deuxième ligne. La ligne 3 crée et affiche une boîte de dialogue avec le titre « Bonjour », l'invite « Quel est votre nom ? », une zone de texte pour que l'utilisateur saisisse son nom, et deux boutons, « OK » et « Annuler ». Une fois que l'utilisateur appuie sur « OK », la réponse est renvoyée dans la variable nomutilisateur. La dernière ligne (pour l'instant) affiche alors : « Bonjour {nom} depuis Python sur Android ! ». Ce n'est pas nouveau, nous l'avons fait la dernière fois. Maintenant, nous allons ajouter plus de code (ci-dessus).

Enregistrez votre code sous « atest1.py ». Nous allons envoyer cela à notre machine virtuelle après avoir discuté de ce qu'il fait.

Jetez un coup d'œil aux quatre premières lignes que nous venons de saisir. Nous créons une boîte de dialogue d'alerte demandant « Voulez-vous jouer à un jeu ? » Dans le cas d'une boîte de dialogue d'alerte, il n'y a pas de zone de texte pour saisir quoi que ce soit. Les deux lignes suivantes permettent de créer deux boutons, l'un avec le texte « Oui », qui est un bouton « positif », et l'autre avec le texte « Non », un bouton « négatif ». Les boutons positif et négatif se réfèrent à la réponse retournée - positive ou négative. La ligne suivante affiche alors la boîte de dialogue. Les sept lignes suivantes attendent une réponse de l'utilisateur.

Nous créons une simple boucle (while True:), puis attendons une réponse pendant 10 secondes en utilisant la fonction droid.eventWait(valeur). La réponse (soit positive soit négative) sera retournée dans - vous l'aurez deviné - la variable reponse. Si cette variable contient « dialog », nous sortons de la boucle et retournons la réponse. Si rien ne se passe avant que le délai soit dépassé, nous sortons simplement de la boucle. L'information réelle retournée dans la variable réponse ressemble à ceci (en supposant qu'on ait choisi le bouton positif, soit « Oui ») :

```
{u'data': {u'which':
u'positive'}, u'name':
u'dialog', u'time':
1339021661398000.0}
```

Vous pouvez voir que la valeur est passée dans le tableau « data », le texte « dialog » dans le tableau « name » et il y a une valeur « time » dont nous ne nous soucions pas ici.

Enfin, nous fermons la boîte de dialogue.

Avant que nous puissions envoyer notre code à la machine virtuelle, il faut la faire démarrer. Exécutez votre émulateur Android. Une fois qu'il a démarré, vous remarquerez que la barre de titre contient quatre chiffres au début du titre. Il s'agit du port sur lequel la machine est à l'écoute. Dans mon cas (et probablement dans le vôtre) il s'agit de 5554.

Maintenant, nous allons envoyer le code vers notre machine virtuelle. Ouvrez une fenêtre de terminal et allez dans le dossier où vous avez enregistré le code. En supposant que vous avez configuré votre chemin par défaut pour y inclure le SDK, saisissez :

```
adb devices
```

Ceci demande à adb de montrer tous les périphériques qui sont connectés. Cela peut inclure non seulement l'émulateur Android, mais aussi les smartphones, tablettes ou autres appareils Android. Vous devriez voir quelque chose comme ceci :

```
Liste des périphériques
connectés
emulator-5554    device
```

Maintenant que nous sommes sûrs que notre appareil est connecté, nous voulons envoyer l'application sur le périphérique. La syntaxe est la suivante :

```
adb push fichier_source
chemin_et_fichier_destination
```

Donc, dans mon cas, ce serait :

```
adb push atest1.py
/sdcard/sl4a/scripts/atest1.py
```

Si tout fonctionne correctement, vous obtiendrez un message plutôt décevant semblable à ceci :

```
11 Ko/s (570 octets en 0.046s)
```

Maintenant, sur l'émulateur Android, démarrez SL4A. Vous devriez voir tous les scripts python et, parmi eux, vous devriez voir atest1.py. Tapez (cliquez) sur « atest1.py » et vous verrez une boîte de dialogue avec 6 icônes. De gauche à droite, on trouve « Exécuter dans une fenêtre de dialogue », « Exécuter en dehors d'une fenêtre », « Éditer », « Enregistrer », « Supprimer », et « Ouvrir dans un éditeur externe ». Pour le moment, tapez (cliquez) sur l'icône la plus à gauche « Exécuter dans une fenêtre de dialogue » afin que vous puissiez voir ce qui se passe.

Vous verrez la première boîte de dialogue vous demandant votre nom. Entrez quelque chose dans la boîte et tapez (cliquez) sur le bouton « Ok ». Vous verrez le message bonjour. Ensuite, vous verrez la boîte de dialogue d'alerte. Tapez (cliquez) sur l'un des boutons pour fermer la boîte de dialogue. Nous ne regardons pas les réponses pour l'instant alors peu importe celle que vous choisissez. Maintenant, nous allons ajouter un peu plus de code (en haut à droite).

Je suis sûr que vous comprendrez que ce morceau de code vérifie simplement la réponse et, si c'est « Aucune » à cause du temps d'attente, nous affichons simplement « Trop tard ». Et si c'est effectivement quelque chose que nous voulons, alors nous assignons les données à la variable rdialog. Maintenant, ajoutez le bout de code suivant (ci-dessous) :

```
if response==None:
    print "Timed out."
else:
    rdialog=response["data"]
```

Ce morceau de code regardera les données transmises par l'événement d'appui sur un bouton. Nous vérifions si la réponse a une valeur « which » et, si c'est le cas, c'est bien un appui légitime sur un bouton. Nous vérifions ensuite si le résultat est la réponse « positive » (bouton « OK »). Si c'est le cas, nous créons une autre boîte de dialogue d'alerte, mais cette fois nous allons ajouter une liste d'éléments parmi lesquels l'utilisateur

```
    if rdialog.has_key("which"):
    result=rdialog["which"]
    if result=="positive":
        droid.dialogCreateAlert("Play a Game","Select a game to play")
        droid.dialogSetItems(['Checkers','Chess','Hangman','Thermal
Nuclear War']) # 0,1,2,3
        droid.dialogShow()
        resp = droid.dialogGetResponse()
```

choisira. Dans ce cas, nous proposons à l'utilisateur de choisir parmi une liste comprenant Dames, Échecs, Pendu et Guerre nucléaire et nous attribuons les valeurs 0 à 3 pour chaque élément. (Est-ce que cela commence à paraître familier ? Oui, ça vient d'un film.) Nous affichons ensuite la boîte de dialogue et attendons une réponse. La partie de la réponse qui nous intéresse est sous la forme d'un tableau. En supposant que l'utilisateur tape (clique) sur Échecs, la réponse résultante nous revient comme ça :

```
Result(id=12,
result={u'item':1},
error=None)
```

Dans ce cas, nous sommes vraiment intéressés par la partie « result » des données renvoyées. La sélection est le n° 1 et est contenue dans la clé « item ». Voici la partie suivante du code (en haut à droite) :

```
if resp.result.has_key("item"):
    sel = resp.result['item']
    if sel == 0:
        droid.makeToast("Enjoy your checkers game")
    elif sel == 1:
        droid.makeToast("I like Chess")
    elif sel == 2:
        droid.makeToast("Want to 'hang around' for a while?")
    else:
        droid.makeToast("The only way to win is not to play...")
```

Ici, nous vérifions pour voir si la réponse contient la clé « item » et, si c'est le cas, nous l'affectons à la variable « sel ». Ensuite nous utilisons une boucle if/elif/else pour vérifier les valeurs et traiter celle qui est sélectionnée. Nous utilisons la fonction droid.makeToast pour afficher notre réponse. Bien sûr, vous pourriez ajouter votre propre code ici. Maintenant, voici la dernière partie du code (au milieu à droite).

Comme vous pouvez le voir, nous répondons simplement aux autres appuis de boutons ici.

Sauvegardez, envoyez, puis exécutez le programme.

Comme vous pouvez le voir, SL4A vous donne la possibilité de faire des applications « graphiques », mais pas complètement. Cela ne devrait toutefois pas vous empêcher d'aller de

```
        elif result=="negative":
            droid.makeToast("Sorry.  See you later.")
    elif rdialog.has_key("canceled"):
        print "Sorry you can't make up your mind."
    else:
        print "unknown response=",response
print "Done"
```

l'avant et de commencer à écrire vos propres programmes pour Android. Ne vous attendez pas à les mettre sur le « marché ». La plupart des gens veulent vraiment des applications complètement graphiques. Nous verrons cela la prochaine fois. Pour plus d'informations sur l'utilisation de SL4A, il suffit de faire une recherche sur Internet et vous trouverez beaucoup de tutoriels et d'autres informations.

Au fait, vous pouvez envoyer votre code directement sur votre smartphone ou votre tablette de la même manière.

Comme d'habitude, le code a été mis en place sur pastebin à http://pastebin.com/BHJWqjCf

Rendez-vous la prochaine fois.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Débuter Python - Partie 35

Greg Walters

Cette fois-ci, nous allons dévier un peu de notre exploration de la programmation Android et regarder un nouvel environnement pour la programmation d'interfaces graphiques appelé Kivy. Vous pouvez aller sur http://kivy.org pour télécharger et installer le paquet, avant d'aller plus loin dans cet article. Les instructions d'installation pour Ubuntu sont ici : http://kivy.org/docs/installation/installation-ubuntu.html.

Tout d'abord, Kivy est une bibliothèque Open Source qui gère les écrans tactiles. Si ce n'est pas encore assez chouette, elle est aussi multiplateforme, ce qui signifie qu'elle fonctionne sous Linux, Windows, Mac OSX, iOS et Android. Maintenant vous comprenez pourquoi nous en parlons. Mais rappelez-vous, la plupart du temps, tout ce que vous codez à l'aide de Kivy peut fonctionner sur n'importe laquelle des plateformes ci-dessus sans recodage.

Avant d'aller trop loin, permettez-moi de faire quelques déclarations. Kivy est TRÈS puissant. Kivy vous procure un nouvel ensemble d'outils pour programmer des interfaces graphiques. Tout cela étant dit, Kivy est également assez compliqué à gérer. Vous êtes limité aux widgets qu'ils ont fournis. En outre, il n'existe pas de designer graphique pour Kivy et vous devez donc faire beaucoup de planification préalable avant d'essayer de faire des trucs compliqués. Rappelez-vous aussi que Kivy est constamment en développement et que les choses peuvent changer rapidement. Jusqu'à présent, je n'ai jamais eu de morceaux de mon code qui aient été cassés par une nouvelle version de Kivy, mais c'est toujours une possibilité.

Plutôt que de nous précipiter et de créer notre propre code ce mois-ci, nous allons examiner quelques-uns des exemples livrés avec Kivy et, le mois prochain, nous préparerons le nôtre.

Une fois que vous avez décompressé Kivy dans son propre dossier, utilisez un terminal et allez dans ce dossier. Le mien est dans /home/greg/Kivy-1.3.0. Maintenant, allez dans le dossier des exemples, puis dans le dossier widgets. Regardons l'exemple accordion_1.py.

Il est très simple, mais montre un widget très chouette. Ci-dessous, voici leur code.

Comme vous pouvez le voir, les trois premières lignes sont des déclarations d'importation. Tout widget que vous utilisez doit être importé et vous devez toujours importer App de kivy.app.

Les huit lignes suivantes sont la classe principale de l'application. La classe est définie, puis une routine appelée build est créée. Vous devrez presque toujours avoir une routine de construction (build) quelque part dans vos programmes Kivy. Ensuite nous définissons un objet racine du widget Accordéon. Ensuite, nous créons cinq éléments AccordionItems et définissons leurs titres. Nous ajoutons ensuite dix étiquettes avec le texte « contenu très grand ». Nous ajoutons ensuite chaque étiquette dans le widget racine (l'accordéon) et, enfin, nous retournons l'objet racine. C'est en substance

```
from kivy.uix.accordion import Accordion, AccordionItem
from kivy.uix.label import Label
from kivy.app import App

class AccordionApp(App):
     def build(self):
          root = Accordion()
          for x in xrange(5):
               item = AccordionItem(title='Title %d' % x)
               item.add_widget(Label(text='Very big content\n' * 10))
               root.add_widget(item)
          return root

if __name__ == '__main__':
     AccordionApp().run()
```

ce qu'affiche l'objet racine dans la fenêtre que crée Kivy. En dernier lieu, nous avons la déclaration « if_name_ » puis exécutons l'application.

Allez-y, exécutez cet exemple pour voir ce qu'il fait.

Dans quelques instants, vous verrez une fenêtre s'ouvrir avec cinq barres verticales à l'intérieur. En cliquant sur une barre, on provoque son ouverture ce qui révèle les dix étiquettes. Bien sûr, chaque barre a le même texte sur les dix étiquettes, mais vous êtes capable de changer cela.

Le widget Accordéon peut être utilisé pour toutes sortes de choses, mais la chose qui me vient toujours à l'esprit est un écran de configuration… chaque barre étant un ensemble distinct de configurations.

Ensuite, nous allons prendre l'exemple textalign.py. Il n'est pas aussi « sexy » que le précédent, mais c'est un bon exemple qui vous fournit des informations importantes pour la suite.

Avant de regarder le code, exécutez le programme.

Vous devriez voir une étiquette en haut de la fenêtre, un ensemble de neuf cases rouges avec du texte dans une grille 3×3 et quatre boutons le long du bas de la fenêtre. Lorsque vous cliquez (appuyez) sur chacun des boutons, l'alignement du texte dans les cases rouges changera. La raison principale pour laquelle il faut examiner cet exemple attentivement est qu'il vous montre comment utiliser et contrôler certains des widgets importants, ainsi que la façon de changer l'alignement de vos widgets, ce qui n'est pas totalement intuitif.

Le code de cet exemple se trouve ci-dessus à droite. Je vais le décortiquer. D'abord le code d'importation (en haut à droite).

Ci-dessous se trouve quelque chose de spécial. Ils ont créé une classe qui ne contient aucun code. Je vais en parler dans quelques minutes :

```
class BoundedLabel(Label):

  pass
```

Ensuite, une classe appelée « Selector » (ci-dessous) est créée :

```
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.gridlayout import GridLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.properties import ObjectProperty
```

```
class Selector(FloatLayout):

app = ObjectProperty(None)
```

Maintenant, la classe Application est créée.

C'est ici que la routine de sélection est créée. Un widget GridLayout est créé (appelée grid ou grille), qui dispose de 3 lignes et 3 colonnes. Cette grille va contenir les neuf cases rouges.

```
for valign in ('bottom',
'middle', 'top'):

for halign in ('left',
'center', 'right'):
```

Ici nous avons deux boucles, l'une extérieure et l'autre intérieure.

```
label = BoundedLabel(text='V:
%s\nH: %s' % (valign,
halign),

size_hint=(None, None),

halign=halign, valign=valign)
```

Dans le code ci-dessus, une instance du widget BoundedLabel est créée, une fois pour chacune des neuf cases rouges. Vous voudrez peut-être m'arrêter là et dire : « Mais attendez ! Il n'y a pas de widget BoundedLabel. Il a juste une déclaration pass. » Eh bien, oui et non. Nous créons une instance d'un widget personnalisé. Comme je le disais un peu plus haut, nous en parlerons plus en détail dans un instant.

Dans le bloc de code (en haut à droite de la page suivante), nous examinons la variable « case », qui est passée à la routine de sélection.

Ici, on retire la grille pour effacer l'écran.

```
class TextAlignApp(App):

     def select(self, case):

     grid = GridLayout(rows=3, cols=3, spacing=10, size_hint=(None, None),
                  pos_hint={'center_x': .5, 'center_y': .5})
```

```
if self.grid:

self.root.remove_widget(self.
grid)
```

La méthode bind ici définit la taille et la grille est ajoutée à l'objet racine.

```
grid.bind(minimum_size=grid.s
etter('size'))

self.grid = grid

self.root.add_widget(grid)
```

Rappelez-vous, dans le dernier exemple j'ai dit que vous devrez presque toujours utiliser une routine de construction. Voici celle de cet exemple. L'objet racine est créé avec un widget FloatLayout. Ensuite (au milieu à droite) nous appelons la classe Selector pour créer un objet de sélection, qui est ensuite ajouté à l'objet racine et on initialise l'affichage en appelant self.select(0).

Enfin, l'application est autorisée à fonctionner.

```
TextAlignApp().run()
```

Maintenant, avant d'aller plus loin, nous devons éclaircir quelques petites choses. Tout d'abord, si vous regardez dans le dossier qui contient le fichier .py, vous remarquerez un autre fichier appelé textalign.kv. Ceci est un fichier spécial utilisé par Kivy pour vous permettre de créer vos propres widgets et règles. Lorsque votre application Kivy démarre, elle cherche dans le même répertoire le fichier d'aide .kv. S'il est présent, elle le charge en premier. Voici le code dans le fichier .kv.

Cette première ligne indique la version minimum de Kivy qui doit être utilisée pour exécuter cette application.

```
#:kivy 1.0
```

Ensuite le widget BoundedLabel est créé. Chacune des cases rouges dans l'application est un BoundedLabel.

Color définit la couleur d'arrière-plan de la boîte à rouge (rgb: 1,0,0). Le widget Rectangle crée un rectangle (vous l'aurez deviné). Lorsque nous appelons le widget BoundedLabel dans le code de l'application, nous passons une étiquette en tant que parent. La taille et la position (ici dans le fichier .kv) sont réglées à la taille et la position de l'étiquette.

Ici (à droite sur la page suivante) le widget de sélection est créé. Ce sont les quatre boutons qui apparaissent au bas de la fenêtre ainsi que l'étiquette sur le haut de la fenêtre.

```
if case == 0:
                    label.text_size = (None, None)
          elif case == 1:
                    label.text_size = (label.width, None)
          elif case == 2:
          label.text_size = (None, label.height)
          else:
          label.text_size = label.size
                    grid.add_widget(label)
```

```
     def build(self):
     self.root = FloatLayout()
          self.selector = Selector(app=self)
     self.root.add_widget(self.selector)
     self.grid = None
          self.select(0)
     return self.root
```

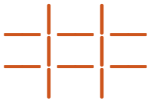
```
<BoundedLabel>:
     canvas.before:
          Color:
               rgb: 1, 0, 0
     Rectangle:
          pos: self.pos
               size: self.size
```

Remarquez que l'étiquette qui constitue le titre en haut de la fenêtre a une position (pos_hint) en haut, a une hauteur de 50 pixels et une taille de police de 16. Chacun des boutons a un texte aligné au centre. La déclaration on_release est une déclaration qui ressemble à « bind », qui fait que, lorsque le bouton est relâché, il appelle (dans ce cas) root.app.select avec la valeur « case ».

J'espère que tout commence à devenir plus clair maintenant. Vous pouvez voir pourquoi Kivy est si puissant.

Examinons un instant deux widgets dont je n'ai pas parlé ci-dessus pendant la discussion du code de l'application, GridLayout et FloatLayout.

GridLayout est un widget parent qui utilise une description en ligne et colonne qui permet de placer les widgets dans chaque cellule. Dans ce cas, il s'agit d'une grille 3×3 (comme un plateau de Tic-Tac-Toe).



Lorsque vous voulez placer un widget dans un GridLayout, vous utilisez la méthode add_widget. Il y a cependant un problème. Vous ne pouvez pas spécifier la cellule de la grille dans laquelle chaque commande sera placée autrement que par l'ordre dans lequel vous les ajoutez. En outre, chaque widget est ajouté de gauche à droite et de haut en bas. Vous ne pouvez pas avoir une cellule vide. Bien sûr, vous pouvez tricher. À vous de comprendre comment.

Le widget FloatLayout semble être juste un conteneur parent pour d'autres widgets enfants.

J'ai passé sous silence quelques points pour l'instant. Cette fois, mon intention était tout simplement de vous enthousiasmer par les possibilités qu'offre Kivy. Dans les prochains articles, nous allons continuer à explorer ce que Kivy peut nous apporter, comment utiliser divers widgets et comment créer un APK pour publier nos applications sur Android.

En atendant, explorez davantage les exemples de Kivy et, surtout, allez sur la page de documentation de Kivy : http://kivy.org/docs/.

```
<Selector>:
      Label:
      pos_hint: {'top': 1}
            size_hint_y: None
      height: 50
            font_size: 16
      text: 'Demonstration of text valign and halign'
      BoxLayout:
            size_hint_y: None
      height: 50
            ToggleButton:
                  halign: 'center'
            group: 'case'
                  text: 'label.text_size =\n(None, None)'
                  on_release: root.app.select(0)
                  state: 'down'
      ToggleButton:
                  halign: 'center'
            group: 'case'
                  text: 'label.text_size =\n(label.width, None)'
                  on_release: root.app.select(1)
      ToggleButton:
                  halign: 'center'
                  group: 'case'
            text: 'label.text_size =\n(None, label.height)'
                  on_release: root.app.select(2)
      ToggleButton:
                  halign: 'center'
                  group: 'case'
            text: 'label.text_size =\n(label.width, label.height)'
                        on_release: root.app.select(3)
```

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

TUTORIEL

Débuter Python - Partie 36

Greg Walters

Avant de commencer, je tiens à souligner que cet article marque les trois ans de la série d'articles sur la programmation Python pour débutants. Je tiens à remercier Ronnie et l'ensemble des personnes du magazine Full Circle pour leur soutien et surtout vous, les lecteurs. Je n'ai jamais pensé que cela continuerait aussi longtemps.

Je tiens aussi à saisir l'occasion de faire une brève remarque au sujet de quelques commentaires flottant dans l'air, suggérant que, après trois ans, le mot « débuter » est peut être déplacé dans le titre de cette série. Après tout, au bout de trois ans, en êtes-vous toujours à débuter ? Eh bien, à certains niveaux, je suis d'accord. Cependant, je reçois encore des commentaires de lecteurs disant qu'ils viennent de découvrir la série et le magazine Full Circle, et qu'ils sont maintenant en train de lire toute la série à partir du début. Ces gens sont donc bien des débutants. Quoi qu'il en soit, à partir de la partie 37, nous enlèverons « débuter » du titre de la série.

Maintenant place au contenu de cet article… la suite sur Kivy.

Imaginez que vous jouez de la guitare. Pas de la « air guitare » (en faisant semblant), mais avec une guitare réelle. Cependant, vous n'êtes pas un très bon joueur de guitare et quelques accords vous posent problème. Par exemple, vous connaissez les accord standards de Do, Mi, Sol, Fa, mais quelques accords – comme Fa# mineur ou Do# mineur – bien que faisables, sont difficiles à faire pendant un morceau rapide. Que faites-vous, surtout si le concert est dans seulement quelques semaines et que vous devez être au point AUJOURD'HUI ? Vous pouvez contourner ce problème en utilisant le capo (cette drôle de pince que vous voyez parfois sur le manche de la guitare). Cela relève la tonalité de la guitare et vous utilisez alors des accords différents pour être comme le reste du groupe. C'est ce qu'on appelle la transposition. Parfois, vous pouvez transposer à la volée dans votre tête. Parfois, il est plus facile de s'asseoir et de travailler sur le papier ; par exemple, si l'accord est Fa# mineur et que vous placez le capo sur la frette 2, vous pouvez simplement jouer un Mi mineur. Mais cela prend du temps. Fabriquons une application qui vous permettra tout simplement de faire défiler la position sur les frettes pour trouver les accords les plus faciles à jouer.

Notre application va être assez simple. Une étiquette de titre, un bouton avec la gamme de base comme texte, une vue défilante « scrollview » (un widget parent merveilleux qui contient d'autres commandes et vous permet de « lancer » ce qu'il contient pour le faire défiler) contenant un certain nombre de boutons qui ont des gammes repositionnées comme texte et un bouton de sortie. Cela ressemblera À PEU PRÈS au texte ci-dessous.

Commencez avec un nouveau fichier Python nommé main.py. Ce nom sera important si/quand vous décidez de créer une application Android avec Kivy. Maintenant, nous allons ajouter nos instructions d'importation qui sont indiquées en haut à droite de la page suivante.

Remarquez la deuxième ligne, « kivy.require('1.0.8') ». Cela vous permet de vous assurer que vous pouvez utiliser les fonctionnalités les plus récentes et les meilleures fournies par Kivy. Notez également que nous incluons une instruction système pour quitter (ligne 3). Plus tard nous aurons un bouton pour quitter.

Voici le début de notre classe appelée « Transpose ».

```
                    Transposer Ver 0.1
-------------------------------------------------------------

 1|  C  C#/Db  D  D#/Eb  E  F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C
 2|  C  C#/Db  D  D#/Eb  E  F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C  C#/Db  D
     D#/Eb  E  F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C  C#/Db D
-------------------------------------------------------------
```

```
class Transpose(App):
    def exit(instance):
        sys.exit()
```

Maintenant, travaillons sur notre routine « build » (au milieu à droite). Elle est nécessaire pour toutes les applications Kivy.

Cela semble confus. Malheureusement, l'éditeur ne garde pas toujours les espaces correctement, même avec une police à espacement fixe. L'idée est que la chaîne texte1 est une simple gamme commençant par la note « Do ». Chacune doit être centrée dans 5 espaces. Comme le texte affiché en bas à droite.

La chaîne texte2 devrait être la même chose, mais répétée. Nous allons utiliser un décalage dans la chaîne texte2 pour remplir le texte du bouton à l'intérieur du widget scrollview.

Nous créons maintenant l'objet racine (qui est notre fenêtre principale) contenant un widget GridLayout. Si vous vous souvenez, il y a TRÈS longtemps, quand nous faisions d'autres développements d'interfaces pour Glade, il y avait un widget grille (« grid view »). Eh bien, le GridLayout ici est à peu près la même chose. Dans notre cas, nous avons une grille qui contient une colonne et trois lignes. Dans chaque cellule de la grille, on peut mettre d'autres widgets. Rappelez-vous, nous ne pouvons pas choisir où va chaque widget autrement que par l'ordre dans lequel on les ajoute.

```
racine =
GridLayout(orientation='verti
cal', spacing=10,
cols=1,rows=3)
```

Dans ce cas, la représentation est la suivante :

```
-----------------------------
(0)   étiquette titre
-----------------------------
(1)   bouton principal
-----------------------------
(2)   scrollview
-----------------------------
```

La vue scrollview contient plusieurs éléments ; dans notre cas ce sont des boutons. Ensuite, on crée l'étiquette qui sera tout en haut de notre application.

```
etiquette =
Label(text='Transposer Ver 0.1',
    font_size=20,
    size_hint=(None,None),
    size=(480,20),
    padding=(10,10))
```

```
import kivy
kivy.require('1.0.8')
from sys import exit
from kivy.app import App
from kivy.core.window import Window
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.scrollview import ScrollView
from kivy.uix.gridlayout import GridLayout
```

Les propriétés qui sont définies devraient être assez explicites. Les seules qui pourraient vous poser problème sont celles de « padding » (remplissage) et de « size_hint ». Le remplissage est le nombre de pixels autour de l'élément dans un repère x,y. On lit dans la documentation Kivy que « size_hint » (pour X, mais c'est identique pour Y) est défini comme suit :

X size hint. Représente la quantité d'espace que le widget doit utiliser selon la direction de l'axe X, par rapport à la largeur de son parent. Seuls Layout et Window utilisent cette propriété. La valeur est indiquée en pourcentage sous forme d'un nombre compris entre 0 et 1, où 1 signifie la taille totale de son parent et 0,5 représente 50 %.

Dans notre cas, size_hint est défini à « none » (aucun), qui vaut par défaut

```
    def build(self):
        #-------------------------------------------------------------
        text1 = "  C  C#/Db  D  D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C"
        text2 = "  C  C#/Db  D  D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C  C#/Db  D
D#/Eb  E    F  F#/Gb  G  G#/Ab  A  A#/Bb  B    C  C#/Db"
        #-------------------------------------------------------------
```

```
|    |    |    |    |    |    |    |    |    |    |    |    |
123456789012345678901234567890123456789012345678901234567890123456
  C  C#/Db  E    F  F#/Gb  G  G#/Ab  A  A#/Bb  B    C
```

100 % ou 1. Ce sera plus important (et compliqué) plus tard.

Maintenant, nous définissons notre bouton « principal » (en haut à droite). Il s'agit d'une référence statique pour la gamme.

Encore une fois, tout devrait être assez clair.

Maintenant, nous ajoutons les widgets à l'objet racine, qui est le widget GridLayout. L'étiquette va dans la première cellule, le bouton (btn1) va dans la seconde.

```
#----------------------------
racine.add_widget(etiquette)
racine.add_widget(btn1)
#----------------------------
```

Arrive maintenant du code plus difficile à comprendre. Nous créons un autre objet GridLayout et l'appelons « s ». Nous le lions ensuite à la hauteur du widget suivant qui, dans ce cas, se trouve être la ScrollView, PAS les boutons.

```
s = GridLayout(cols=1,
spacing = 10, size_hint_y =
None)
s.bind(minimum_height=s.sette
r('height'))
```

Maintenant (au milieu à droite), nous créons 20 boutons, remplissons la propriété « texte », puis les ajoutons au GridLayout.

Maintenant nous créons le ScrollView, définissons sa taille, et l'ajoutons au Grid-Layout racine.

```
sv =
ScrollView(size_hint=(None,
None), size=(600,400))

sv.center =
Window.center

racine.add_widget(sv)
```

Enfin, nous ajoutons le GridLayout, qui contient tous nos boutons dans le ScrollView, et retournons l'objet racine à l'application.

```
sv.add_widget(s)

return racine
```

Enfin, nous avons notre routine « if_name_ ». Remarquez que nous nous réservons la possibilité d'utiliser l'application comme une application android.

```
if __name__ in
('__main__','__android__'):

    Transpose().run()
```

Maintenant, vous vous demandez peut-être pourquoi j'ai utilisé des boutons au lieu d'étiquettes pour tous nos objets textuels. C'est parce que les étiquettes dans Kivy n'ont aucune sorte de bordure visible par défaut. Nous jouerons avec cela dans le prochain épisode. Nous allons également ajouter un bouton pour quitter et d'autres petites choses.

```
    btn1 = Button(text = "  " + text1,size=(680,40),
           size_hint=(None, None),
           halign='left',
           font_name='data/fonts/DroidSansMono.ttf',
           padding=(20,20))
```

```
for i in range(0,19):
    if i <= 12:
        if i < 10:
            t1 = " " + str(i) + "| "
        else:
            t1 = str(i) + "| "
    else:
        t1 = ''
        text2 = ''
    btn = Button(text = t1 + text2[(i*5):(i*5)+65],
          size=(680, 40),
          size_hint=(None,None),
          halign='left',
          font_name='data/fonts/DroidSansMono.ttf')
    s.add_widget(btn)
#--------------------------------------------------------------
```

Le code source peut être trouvé sur Pastebin :

http://pastebin.com/8jTJSmLR

Jusqu'à la prochaine fois, amusez-vous et je vous remercie de m'avoir suivi pendant trois ans !

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

TUTORIEL

Programmer en Python - Partie 37

Greg Walters

Ce mois-ci, nous allons terminer le programme de transposition que nous avons écrit dans Kivy. J'espère que vous avez enregistré le code de la dernière fois, parce que nous allons le compléter. Sinon, récupérez-le sur le FCM n° 64.

Commençons par récapituler ce que nous avons fait le mois dernier. Nous avons créé une application qui permet à un guitariste de transposer rapidement d'une clé à une autre. Le but ultime est de pouvoir exécuter cette application non seulement sous Linux ou Windows, mais également sur un appareil Android. Je l'emporte sur ma tablette quand je vais répéter avec mon groupe. Je me préparais à conditionner notre projet pour Android, mais certaines choses ayant changé dans la méthode pour le faire, nous verrons cela le mois prochain.

L'application, telle que nous l'avons laissée la dernière fois, ressemblait à ce qui est ci-dessous à gauche.

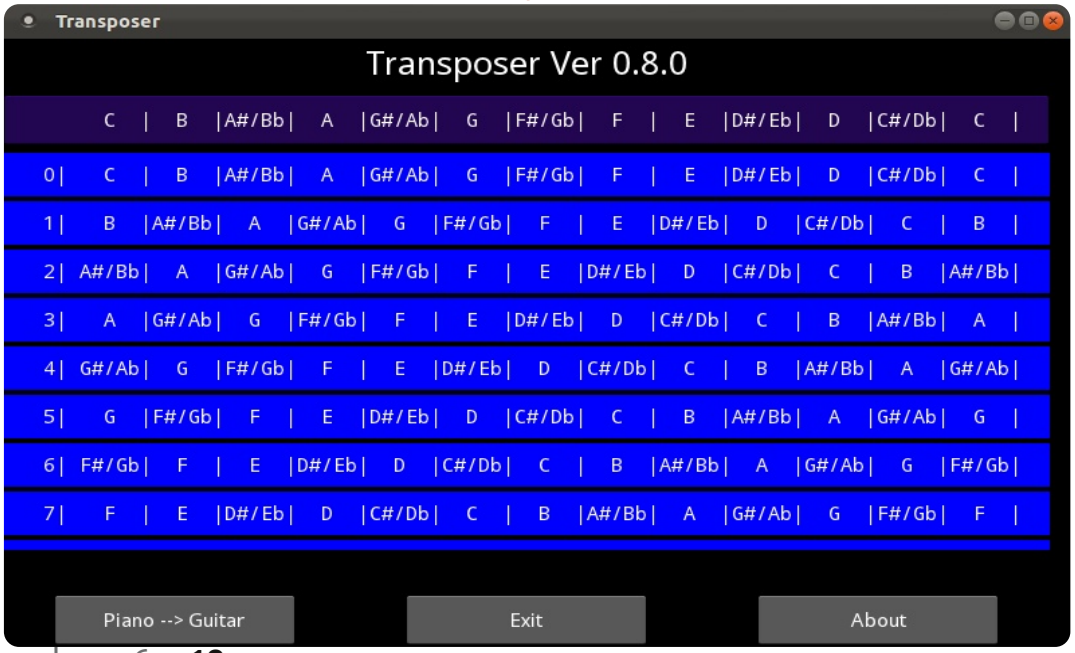
Lorsque nous aurons terminé, elle devrait ressembler à l'écran ci-dessous à droite.

La première chose que vous remarquerez est qu'il y a des étiquettes bleues à la place de celles qui étaient grises et tristes. La suivante est qu'il y a trois boutons. Enfin, les étiquettes qui défilent sont plus proches de la largeur totale de la fenêtre. À part ça, c'est à peu près la même chose (visuellement). L'un des boutons est un bouton « à propos » qui affichera quelques informations simples, pour vous montrer comment faire un pop-up simple. Un autre bouton sert à quitter. Le troisième bouton va remplacer l'étiquette de texte pour faciliter la transposition de piano à guitare ou vice-versa.

```
#:kivy 1.0
#:import kivy kivy

<BoundedLabel>:
      canvas.before:
      Color:
            rgb: 0, 0, 1
      Rectangle:
            pos: self.pos
            size: self.size
```

Nous allons commencer par créer un fichier .kv (ci-dessus). C'est ce qui va nous donner les étiquettes colorées. C'est un fichier très simple.

Les deux premières lignes sont né-



cessaires. Elles indiquent simplement quelle version de Kivy est requise. Ensuite, nous créons un nouveau type d'étiquette appelé « BoundedLabel ». La couleur est réglée avec des valeurs RVB (entre 0 et 1, ce dernier représentant 100 %), et comme vous pouvez le voir, la valeur de bleu est fixée à 100 pour cent. Nous allons également créer un rectangle qui est l'étiquette réelle. Enregistrez ce fichier sous le nom « transpose.kv ». Vous devez prendre le même nom que la classe qui va l'utiliser.

Maintenant que ceci est terminé, ajoutez les lignes suivantes juste avant la classe transpose dans le fichier source de la dernière fois :

```
class BoundedLabel(Label):

    pass
```

Pour que cela fonctionne, il suffit d'une définition. Avant d'aller plus loin, ajoutez la ligne suivante à la section des importations :

```
from kivy.uix.popup import
Popup
```

Cela nous permettra de créer le popup plus tard. Maintenant, dans la classe Transpose, juste à l'intérieur de la routine build, placez le code ci-dessus à droite.

```
def ChargeEtiquettes(w):
    if w == 0:
            tex0 = self.texte1
            tex1 = self.texte2
        else:
            tex0 = self.texte3
            tex1 = self.texte4
        for i in range(0,22):
            if i <= 12:
               if i < 10:
                  t1 = " " + str(i) + "| "
               else:
                  t1 = str(i) + "| "
                  t = tex1
            else:
                t1 = ''
                t = ''
            l = BoundedLabel(text=t1+t[(i*6):(i*6)+78], size=(780, 35),
                size_hint=(None, None),halign='left',
                font_name='data/fonts/DroidSansMono.ttf')
            s.add_widget(l)
```

La routine ChargeEtiquettes nous donnera les étiquettes de couleur (BoundedLabel) et la capacité d'échange. Vous avez pratiquement tout vu la dernière fois. Nous passons une valeur au paramètre « w » pour déterminer quel texte est affiché. La ligne l=BoundedLabel est à peu près la même que la dernière fois, sauf que, cette fois, nous utilisons un widget BoundedLabel au lieu d'un widget Bouton. Les « ChargeEtiquettes » seront principalement appelés depuis la routine suivante, Echange. Placez ce code (à droite) en dessous de ChargeEtiquettes.

```
def Echange(instance):
    if self.quelsens == 0:
        self.quelsens = 1
        btnSens.text = "Guitare --> Piano"
        btn1.text = "    " + self.texte3
        s.clear_widgets()
        ChargeEtiquettes(1)
    else:
        self.quelsens = 0
        btnSens.text = "Piano --> Guitare"
        btn1.text = "    " + self.texte1
        s.clear_widgets()
        ChargeEtiquettes(0)
```

```
self.quelsens=0
self.texte1 = "  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Eb|  D  |C#/Db|  C  |"
self.texte2 = "  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Eb|  D  |C#/Db|  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Ab|  D  |C#/Db|  C  |"
self.texte3 = "  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |"
self.texte4 = "  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |C#/Db|"
```

Vous pouvez voir que cette routine est assez explicite. Nous utilisons une variable (self.quelsens) pour déterminer « dans quel sens » les étiquettes s'affichent… de Guitare vers Piano ou de Piano vers Guitare.

Assurez-vous de sauvegarder votre travail dès à présent, car nous allons faire beaucoup de changements à partir de maintenant.

Remplacez les lignes définissant texte1 et texte2 par les lignes ci-dessus (tableau en haut de page).

Nous réglons self.quelsens à 0 qui sera notre valeur par défaut pour la procédure d'échange. Ensuite, nous définissons quatre chaînes au lieu des deux que nous avions la dernière fois. Vous remarquerez peut-être que les chaînes texte3 et texte4 sont en fait texte1 et texte2 à l'envers.

Maintenant, nous allons adapter la définition de la ligne racine. Changez-la de :

```
root =
GridLayout(orientation='verti
cal', spacing=10,
cols=1,rows=3)
```

à :

```
root =
GridLayout(orientation='verti
cal', spacing=6, cols=1,
rows=4,
row_default_height=40)
```

Nous avons changé l'espacement de 10 à 6 et réglé la hauteur de ligne par défaut à 40 pixels. Changez le texte de l'étiquette (ligne suivante) en « text='Transposer Ver 0.8.0' ». Pour le reste, rien n'a changé sur cette ligne.

Maintenant changez la définition du bouton de :

```
btn1 = Button(text = "   " +
text1,size=(680,40),

  size_hint=(None,None),

  halign='left',

font_name='data/fonts/DroidSa
nsMono.ttf',

  padding=(20,20))
```

à :

```
btn1 = Button(text = "      "
+ self.text1,size=(780,20),

  size_hint=(None, None),

  halign='left',

font_name='data/fonts/DroidSa
nsMono.ttf',

  padding=(20,2),

background_color=[0.39,0.07,.
92,1])
```

Remarquez que j'ai changé le format de la première définition pour plus de clarté. Les gros changements sont la taille qui passe de 680,40 à 780,20 et la couleur de fond du bouton. Rappelez-vous, on peut changer la couleur de fond pour les boutons, mais pas pour les étiquettes « standards ».

Ensuite, nous allons définir trois widgets AnchorLayout pour les trois boutons que nous ajouterons plus tard. Je les ai nommés al0 (AnchorLayout0), al1 et al2. Nous ajoutons également le code pour le Popup « à propos » et définissons nos boutons avec les paramètres de liaison (bind). Ceci est illustré à la page suivante, en haut à gauche.

Trouvez la ligne « s = GridLayout » et modifiez l'espacement de 10 à 4. Ensuite, ajoutez la ligne suivante après la ligne s.bind (juste avant la boucle for) :

```
ChargeEtiquettes(0)
```

Ceci appelle la routine ChargeEtiquettes avec notre « quelsens » par défaut qui vaut 0.

Ensuite, commentez la totalité du code de la boucle for. Cela commence par « for i in range(0,19): » et se termine par « s.add_widget(btn) ». Nous n'avons pas besoin de cette routine puisque ChargeEtiquettes le fait pour nous.

Maintenant, enregistrez votre code

```
al0 = AnchorLayout()
al1 = AnchorLayout()
al2 = AnchorLayout()
popup = Popup(title='A propos de Transposer',
        content=Label(text='Ecrit par G.D. Walters'),
        size_hint=(None,None),size=(400,400))
btnSens = Button(text = "Piano --> Guitare",
         size=(180,40),size_hint=(None,None))
btnSens.bind(on_release=Echange)
btnAPropos=Button(text="A propos",
         size=(180,40),size_hint=(None,None))
btnAPropos.bind(on_release=AfficheAPropos)
btnQuitter =Button(text="Quitter",
         size=(180,40),size_hint=(None,None))
btnQuitter.bind(on_release=exit)
```

et essayez de l'exécuter. Vous devriez voir un bouton violet foncé en haut et nos BoundLabels d'un joli bleu. De plus, vous remarquerez que les BoundLabels dans la fenêtre de défilement sont plus rapprochés, ce qui en facilite grandement la lecture.

Nous sommes presque au bout de notre code, mais il nous reste quelques petites choses à faire. Après la ligne « sv = ScrollView », ajoutez la ligne suivante :

```
sv.size = (720, 320)
```

Cela définit la taille du widget ScrollView à 720 sur 320, ce qui le rend plus large à l'intérieur de la fenêtre racine. Maintenant, avant la ligne « return racine », ajoutez le code que vous voyez en haut à droite.

Ici, nous ajoutons les trois boutons aux widgets AnchorLayout, créons un GridLayout pour contenir les AnchorLayouts et enfin ajoutons les AnchorLayouts au GridLayout.

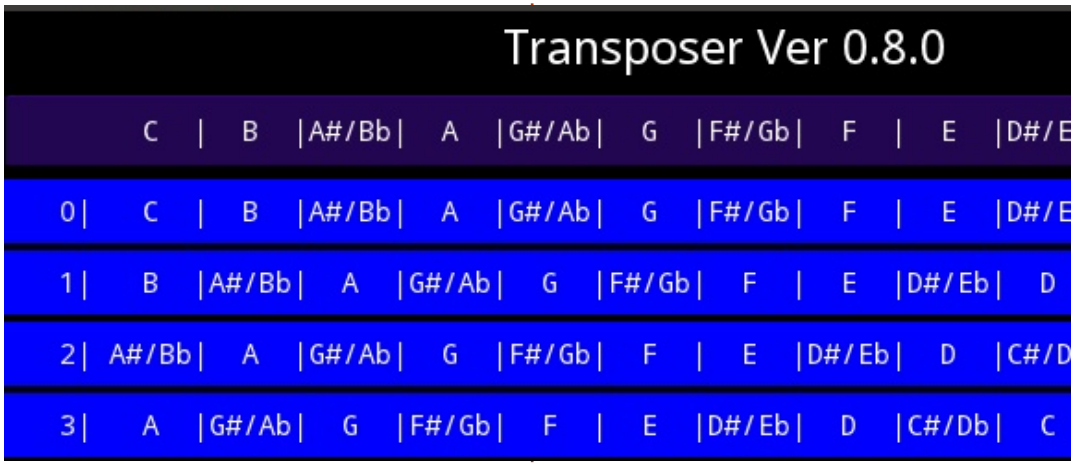
Retournez juste en dessous de la routine « def Echange » et ajoutez ce qui suit :

```
def AfficheAPropos(instance):

  popup.open()
```

C'est tout. Enregistrez et exécutez le code. Si vous cliquez sur le bouton « À propos », vous verrez le popup tout simple. Cliquez n'importe où en dehors du popup pour le faire disparaître.

Maintenant, notre code est écrit. Vous pouvez trouver le code complet

```
        al0.add_widget(btnSens)
        al1.add_widget(btnQuitter)
        al2.add_widget(btnAPropos)
        bgl = GridLayout(orientation='vertical',
              spacing=6, cols=3,rows=1,
              row_default_height=40)
        bgl.add_widget(al0)
        bgl.add_widget(al1)
        bgl.add_widget(al2)
```



ici : http://pastebin.com/T0kJ0q5z

Ensuite, nous devons créer notre paquet Android… mais cela devra attendre la prochaine fois.

Si vous voulez vous préparer et essayer d'empaqueter pour Android avant le mois prochain, allez sur http://kivy.org/docs/guide/packaging-android.html pour trouver la documentation à ce sujet. Assurez-vous de suivre attentivement la documentation.

Rendez-vous le mois prochain.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

TUTORIEL

Écrit par Greg Walters

Programmer en Python : Partie 38

Comme je l'ai promis la dernière fois, nous allons reprendre l'application de transposition que nous avons écrite et créer un APK pour l'installer sur votre appareil Android.

Avant de commencer, assurons-nous que tout soit prêt. La première chose dont nous avons besoin est de placer les deux fichiers que nous avons créés la dernière fois dans un dossier auquel vous pouvez facilement accéder. Appelons-le « transposer ». Créez-le dans votre répertoire personnel. Ensuite, copiez les deux fichiers (transpose.kv et transpose.py) dans ce dossier. Maintenant renommez transpose.py en main.py. Cette partie est importante.

Ensuite, nous avons besoin de faire référence aux instructions d'empaquetage de Kivy dans un navigateur Web. Le lien est http://kivy.org/docs/guide/packaging-android.html. Nous allons utiliser ceci pour les prochaines étapes, mais pas exactement comme les gens de Kivy l'ont prévu. Vous devez avoir le SDK android de notre leçon précédente. Idéalement, vous devriez y retourner et récupérer tous les logiciels qui y sont listées, mais pour nos besoins, il vous suffit de suivre ce qui est indiqué ici. Vous aurez besoin de télécharger le logiciel python-for-android. Ouvrez une fenêtre de terminal et tapez ce qui suit :

```
git clone
git://github.com/kivy/python-
for-android
```

Ceci va télécharger et installer le logiciel dont nous avons besoin pour continuer. Maintenant, dans un terminal, allez dans le répertoire du dossier python-for-android/dist/default.

Vous allez y trouver un fichier appelé build.py. C'est lui qui va faire tout le travail pour nous. Et maintenant, voici la magie.

Le programme build.py prend divers arguments sur la ligne de commande et créera l'APK pour vous. Ci-dessus se trouve la syntaxe pour build.py prise directement dans la documentation Kivy.

```
./build.py --dir <path to your app>
           --name "<title>"
           --package <org.of.your.app>
           --version <human version>
           --icon <path to an icon to use>
           --orientation <landscape|portrait>
           --permission <android permission like VIBRATE> (multiple allowed)
           <debug|release> <installd|installr|...>
```

Pour nos besoins, nous allons utiliser la commande suivante (le « \ » est un caractère de continuation de ligne) :

```
./build.py --dir ~/transposer
--package
org.RainyDay.transposer \
 --name "RainyDay Transposer"
--version 1.0.0 debug
```

Regardons les morceaux de la commande :

./build.py - c'est l'application ;
--dir ~/transposer - il s'agit du répertoire contenant le code de l'application ;
--package org.RainyDay.transposer - c'est le nom du paquet ;
--name "RainyDay Transposer" - c'est le nom de l'application qui apparaîtra dans la liste des applications ;
--version 1.0.0 - la version de notre application ;
debug - c'est le niveau de sortie (debug ou release).

Une fois que vous aurez exécuté ceci, en supposant que tout a fonctionné comme prévu, vous devriez avoir un certain nombre de fichiers dans le dossier /bin. Celui que vous cherchez est intitulé « RainyDay-Transposer-1.0.0-debug.apk ». Vous pouvez le copier sur votre appareil Android en utilisant votre gestionnaire de fichiers favori, puis l'installer comme n'importe quelle autre application des divers magasins d'applications.

C'est tout le temps dont je dispose pour ce mois-ci.