Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIES PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIES PROGRAMMATION

PROGRAMMER EN PYTHON

Premier volume

About Full Circle

Full Circle is a free, independent, magazine dedicated to the Ubuntu family of Linux operating systems. Each month, it contains helpful how-to articles and reader- submitted stories. Full Circle also features a companion podcast, the Full Circle Podcast which covers the magazine, along with other news of interest.

Please note: this Special Edition is provided with absolutely no warranty whatsoever; neither the contributors nor Full Circle Magazine accept any responsibility or liability for loss or damage resulting from readers choosing to apply this content to theirs or others computers and equipment.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Welcome to another 'single-topic special'

In response to reader requests, we are assembling the content of some of our serialised articles into dedicated editions.

For now, this is a straight reprint of the series 'Programming in Python', Parts 22-26 from issues #48 through #52; nothing fancy, just the facts.

Please bear in mind the original publication date; current versions of hardware and software may differ from those illustrated, so check your hardware and software versions before attempting to emulate the tutorials in these special editions. You may have later versions of software installed or available in your distributions' repositories.

Enjoy!

Find Us

Website:
http://www.fullcirclemagazine.org/

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

IRC: #fullcirclemagazine on chat.freenode.net

Editorial Team
Editor: Ronnie Tucker
(aka: RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster: Rob Kerfia
(aka: admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcaster: Robin Catling
(aka RobinCatling)
podcast@fullcirclemagazine.org
Comm. Manager: 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'exem-ple 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 Jours DansMois[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 « jours-dansmois » 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, intéres-sons 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 vou- lons 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 une valeur à une variable, on utilise s ce qui se trouve dans le bloc indenté en dessous sera exécuté. La plupart du temps cela sera sufisant 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 pou-vez 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 n'est 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 instructionsif/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 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, joueez 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

Programmer en Python - Partie 3

Greg Walters

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 (ci-dessus 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 « HautOu Bas » 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. Enregistrez le code dans le fichier « pprint1.py » et exécutez-le. La sortie devrait ressembler à l'image ci-dessus, 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],['B
onbons',.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 (celle se terminent par #NOUVEAU) pour que votre code ressemble à celui en haut à droite. 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 ci-dessous. 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

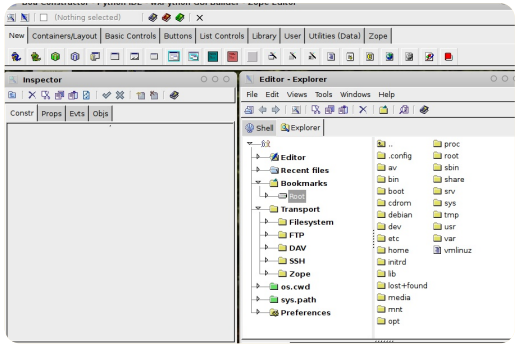
Programmer en Python - Partie 5

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.

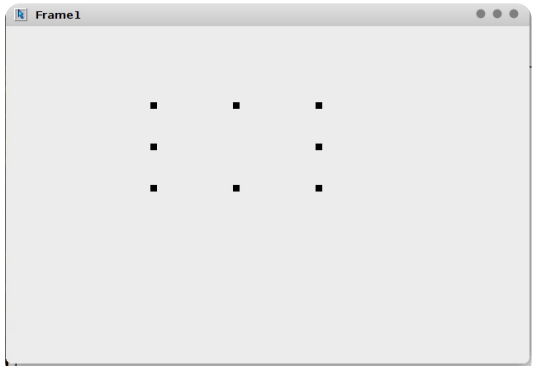
Ok, allons-y. Dans l'onglet Nouveau de la palette des outils, sélectionnez « wx.App » (5e bouton en partant de

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 :

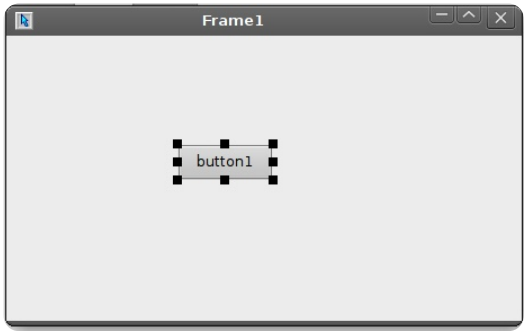


Vous vous rappelez quand je vous ai parlé des bogues ? Eh bien, en voici un. Ne vous inquiétez pas. Vous apercevez les 8 petits carrés noirs ? Ils représentent les bords du panneau. Si vous voulez, vous pouvez cliquer et faire glisser l'un d'eux pour redimensionner le panneau. Ce qui nous intéresse cette 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.

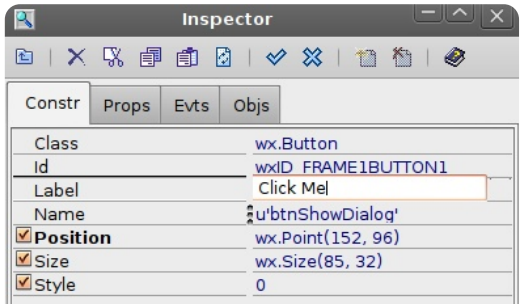
Retournez dans le mode Conception. Cliquez sur l'onglet « Boutons » dans la palette des outils, puis cliquez sur le premier bouton de gauche, le bouton « wx.Button ». Positionnez-le quelque part près du centre de votre cadre. Vous devez obtenir quelque chose comme ceci :



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,

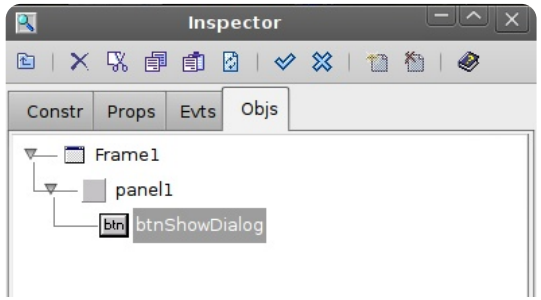
appuyez sur la touche Contrôle (Ctrl) et tout en la laissant enfoncée, utilisez les flèches du clavier pour le déplacer à votre guise.

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



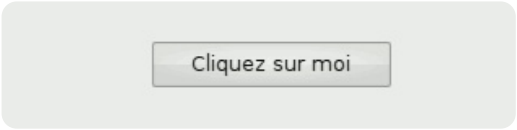
le nom en « btnAfficheDialog » et l'étiquette en « Cliquez sur moi ».

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



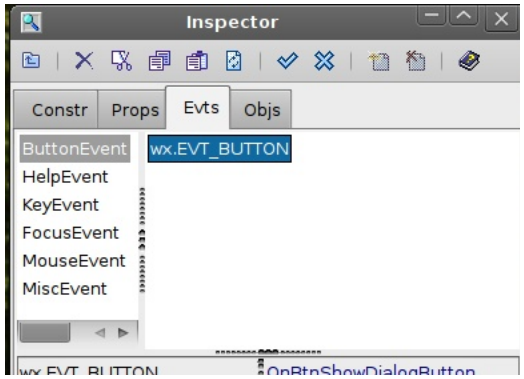
relation parent/enfant. Comme vous pouvez le voir, le bouton est un enfant du « panel1 », qui est un enfant du « Frame1 ».

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. En-suite, 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 : « OnbtnAf-ficheDialogButton » 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.

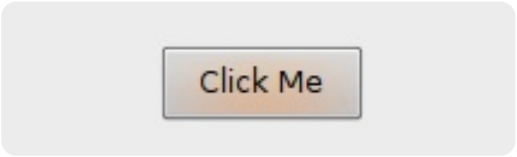
Ensuite, regardez la fonction «___init___». Vous pouvez placer ici n'importe quel appel à du code d'initialisation. Enfin, le bloc OnBtnAf-ficheDialogButton. C'est là que nous plaçons le code qui fera tout le travail lorsque l'utilisateur cliquera sur le bouton. Remarquez qu'il y a une ligne event.Skip() 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 - Affi-cher une icône erreur.

wx.ICON_INFORMATION - Afficher une icône in-formation.

La façon d'écrire cela pourrait être :

```
wx.MessageBox('Vous 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

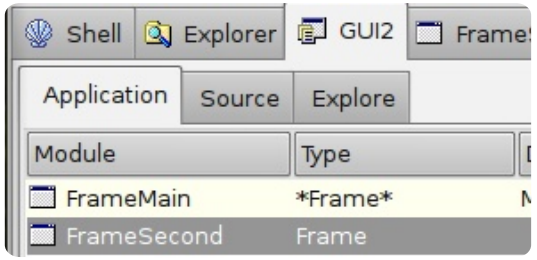
Greg Walters

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).

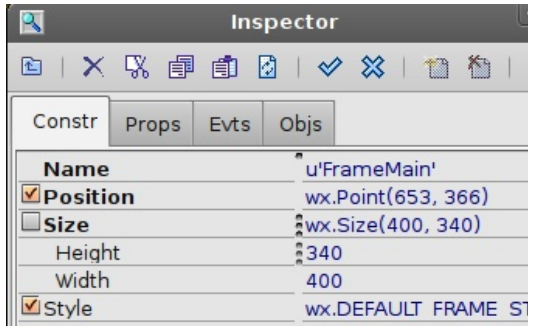
Avant de faire quoi que ce soit, enregistrez Frame1 sous le nom « FenetrePrincipale.py » puis enregistrez App1 sous le nom « GUI2.py ». C'est important. Après avoir sélectionné l'onglet GUI2 dans l'éditeur, allez dans la palette d'outils et ajoutez une autre fenêtre à votre projet en cliquant sur wx.Frame (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.



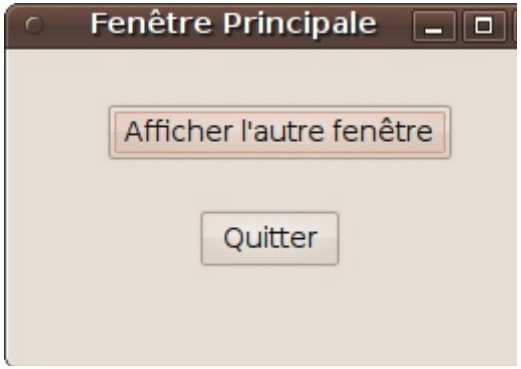
Maintenant cliquez sur l'onglet props. Cliquez sur la propriété « Centered » et paramétrez-la sur wx.BOTH. Cliquez sur l'icône Envoyez et enregistrez votre travail. Maintenant, lancez l'application en cliquant sur l'icône en forme de flèche jaune. Notre programme affiche une fenêtre au centre de l'écran 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



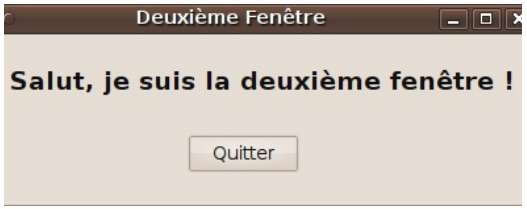
au-dessus de l'autre et près du centre de la fenêtre. Sélectionnez le bouton du haut et nommez-le « btnAfficheNouveau » puis saisissez son étiquette (Label) « Afficher l'autre fenêtre » dans l'onglet constr de l'inspecteur. Utilisez la combinaison Maj+Flèche pour redimensionner le bouton afin que tout 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()
```

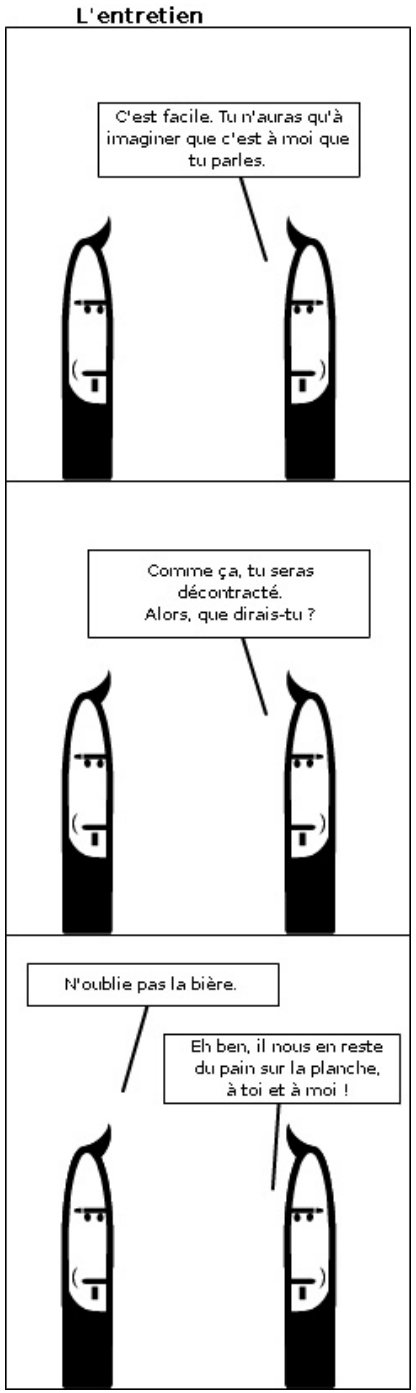
FrameSecond code:

```
#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'il 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 Ingredients 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 :

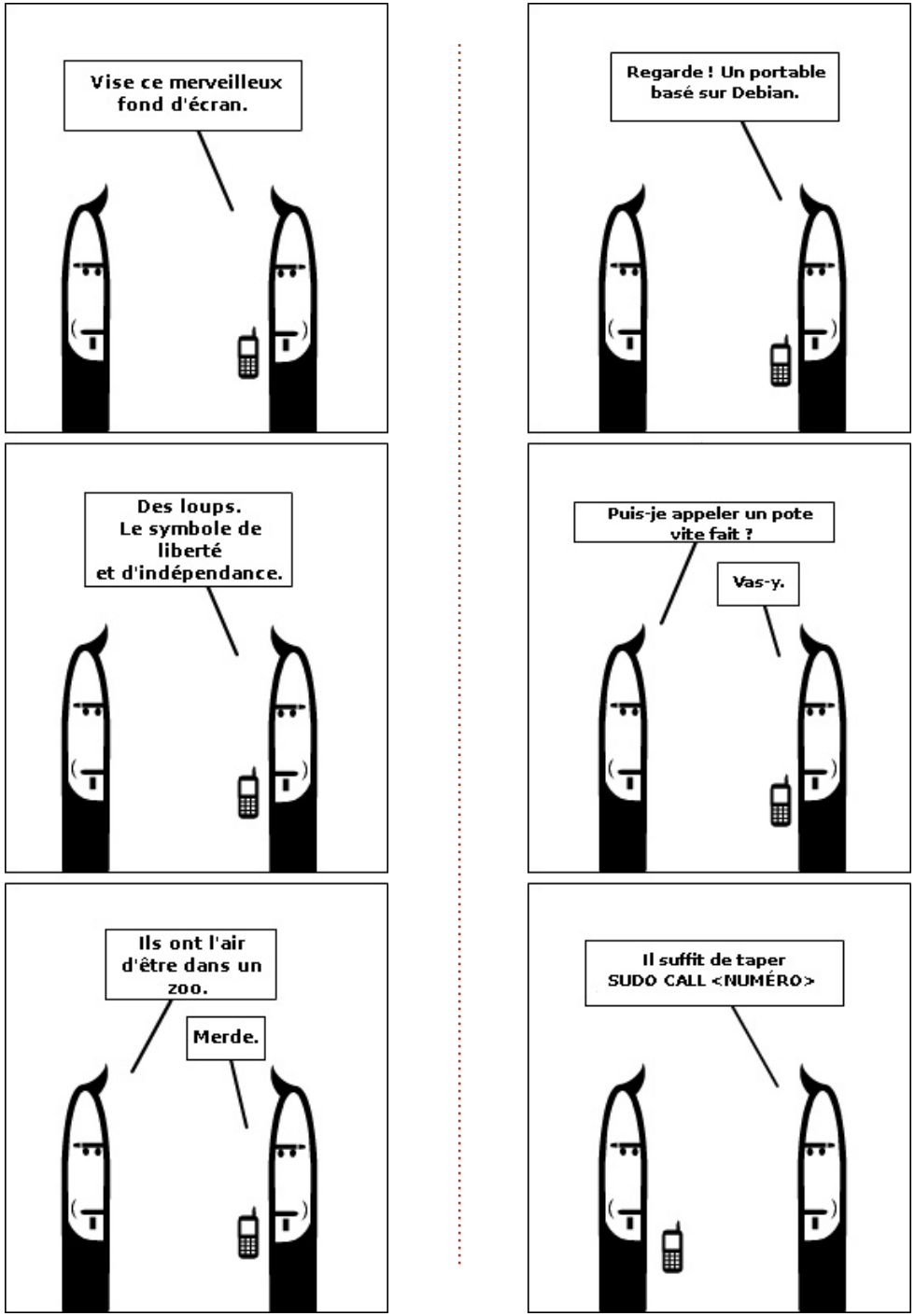
```
sql = 'INSERT INTO
Ingredients
(idRecette,ingredients)
VALUES ( %s,"1 tasse de riz
étuvé (cru)")' % dernierID
curseur.execute(sql)

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() attendre le choix de pour 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 ci-dessus 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 ci-dessous 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 uneseulerecette). 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 12.

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 13.

```
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 tablerecette 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 webbrowser.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"
```