ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

full circle magazine n'est affilié en aucune manière à Canonical Ltd

## Au sujet du Full Circle

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

## Clause de non-responsabilité :

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

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

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

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

**Amusez-vous !**

## Nos coordonnées

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

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

**IRC :** #fullcirclemagazine on chat.freenode.net

**Équipe éditoriale :**

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

Les articles contenus dans ce magazine sont publiés sous la licence Creative Commons Attribution-Share Alike 3.0 Unported license. Cela signifie que vous pouvez adapter, copier, distribuer et transmettre les articles mais uniquement sous les conditions suivantes : vous devez citer le nom de l'auteur d'une certaine manière (au moins un nom, une adresse e-mail ou une URL) et le nom du magazine (« Full Circle Magazine ») ainsi que l'URL www.fullcirclemagazine.org (sans pour autant suggérer qu'ils approuvent votre utilisation de l'œuvre). Si vous modifiez, transformez ou adaptez cette création, vous devez distribuer la création qui en résulte sous la même licence ou une similaire.

**Full Circle Magazine est entièrement indépendant de Canonical, le sponsor des projets Ubuntu. Vous ne devez en aucun cas présumer que les avis et les opinions exprimés ici aient reçus l'approbation de Canonical.**

# TUTORIEL

Greg Walters

# Programmer en Python - Partie 9

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

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

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

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

Maintenant que Mutagen installé, commençons à coder.

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

```
from mutagen.mp3 import MP3

import os

from os.path import
join,getsize,exists

import sys

import apsw
```

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

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

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

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

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

Qu'est-ce que c'est que ça ?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## MON HISTOIRE RAPIDE

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

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

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

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

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

**Brian G Hartnell** - *Photographe*

# TUTORIEL

Greg Walters

# Programmer en Python - Partie 10

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

• de vous familiariser avec ce qu'est XML.

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

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

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

Regardons un exemple de fichier XML générique :

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

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

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

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

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

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

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

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

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

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

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

```
import elementtree.ElementTree as ET
arbre = ET.parse('xmlexemple1.xml')

ET.dump(arbre)
```

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

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

Maintenant, remplacez le code par ce qui suit :

```
import elementtree.ElementTree as ET

arbre = ET.parse('xmlexemple1.xml')

individu = arbre.findall('.//individu')

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

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

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

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

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

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

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

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

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

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

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

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

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

Dossier Navicache

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

```
import elementtree.Element-
Tree as ET

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

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

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

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

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

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

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

Open Cache: Unrestricted

Cache Type: Normal

Cache Size: Normal

Difficulty: 1.5

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

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

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

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

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

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

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

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

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

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

**Amusez-vous bien !**

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

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

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

print "terminé"
```

# TUTORIEL

Greg Walters

# Programmer en Python - Partie 11

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

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

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

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

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

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

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

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

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

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

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

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

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

Tout d'abord, nous vérifions ce

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

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

```
from xml.etree import Element-
Tree as ET

import urllib

import sys

import getopt
```

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

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

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

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

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

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

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

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

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

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

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

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

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

```
def DoIt(self,Localisation):

self.getCurrents(1,Localisation)

self.output()
```

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

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

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

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

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

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

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

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

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

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

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

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

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

Ici, on affiche simplement les variables.

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

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

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

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

getCurrents routine

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

**D'ici là, amusez-vous bien !**

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


# EXTRA ! EXTRA ! LISEZ CECI

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

**LE SERVEUR PARFAIT ÉDITION SPECIALE**

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

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

**PYTHON ÉDITION SPECIALE n° 1**

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

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

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

# TUTORIEL

Greg Walters

# Programmer en Python - Partie 12

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Amusez-vous bien jusqu'à la prochaine fois.**

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

Les thèmes de cet épisode sont :
- Actus - développement de Maverick
- Entrevue Lubuntu
- Jeux - Ed critique Osmos
- Retours

... et toute la bonne humeur habituelle.

**Vos animateurs :**
*Robin Catling*
*Ed Hewitt*
*Ronnie Tucker*

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

# Programmer en Python - Partie 13

Greg Walters

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

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

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

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

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

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

Exemple très simple, si ce n'est le

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

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

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

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

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

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

```
#!/usr/bin/env python
# CursesExample2
import curses
#==========================================================
# BOUCLE PRINCIPALE
#==========================================================
try:
    monEcran = curses.initscr()
    monEcran.clear()
    monEcran.addstr(0,0,"0         1         2         3
        4         5         6         7")
monEcran.addstr(1,0,"1234567890123456789012345678901234567
8901234567890123456789012345678901234567890")
    monEcran.addstr(10,0,"10")
    monEcran.addstr(20,0,"20")
    monEcran.addstr(23,0, "23 - Appuyez sur une touche
pour continuer")
    monEcran.refresh()
    monEcran.getch()
finally:
    curses.endwin()
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**À la prochaine fois.**

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

Les thèmes de cet épisode sont :
- Actualités.
- Opinion : contribuer à des articles avec le rédacteur en chef de FCM.
- Interview avec Amber Graner.
- Retours.

... et toute la bonne humeur habituelle.

**Vos animateurs :**
*Robin Catling*
*Ed Hewitt*
*Ronnie Tucker*

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

# TUTORIEL

Greg Walters

# Programmer en Python - Partie 14

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

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

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

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

```
myscreen.addstr([ligne],[col
onne],[texte],curses.color_p
air(X))
```

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

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

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

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

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

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

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

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

Bien que cela ait l'air assez simple, coder ce jeu nécessite quand même beaucoup de travail.

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

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

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

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

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

```
import curses
import random

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Nous aurons également besoin de créer une routine qui initialise chaque

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

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

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

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

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

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

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

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

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

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

            saisie = ecran.getch(14,56)
            if saisie == ord("N") or saisie == ord("n"):
                break
            else:
                ecran.clear()
```

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

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

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

```
    def AfficheScore(self,ecran):

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# TUTORIEL — Programmer en Python - Partie 15

Greg Walters

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

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

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

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

Juste après les instructions d'importation, ajoutez :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

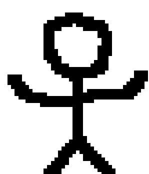
```
import pygame
from pygame.locals import *
import os
print
pygame.font.match_font('Couri
er New')
```

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

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

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

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



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

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

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

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

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

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

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

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

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

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

(...)

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

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

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

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

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

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

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

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

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

# TUTORIEL — Programmer en Python - Partie 16

Greg Walters

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

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

C'est parti.

## La commande PRINT

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

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

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

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

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

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

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

## Mise en forme et substitution de variable

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

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

AVANT

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

APRÈS

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

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

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

>>>
```

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

## Les nombres

Sous Python 2.x, si on faisait :

```
x = 5/2.0
```

x contenait 2.5. Mais si on faisait :

```
x = 5/2
```

x contenait 2 à cause de l'arrondi.

Sous 3.x, si on fait :

```
x = 5/2
```

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

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

## Les saisies

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

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

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

```
Traceback (most recent call
last):

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

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

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

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

et tout fonctionne correctement.

## Non égalité

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

## Convertir d'anciens programmes en Python 3.x

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Est-ce que je passe tout de suite à 3.x ?

La plupart des problèmes sont les mêmes dans tout changement d'un langage de programmation. Les changements de syntaxe sont nombreux avec chaque nouvelle version. Des raccourcis comme += ou -= apparaissent à l'improvisten et, en fait, nous facilitent la vie.

Quels sont les inconvénients à migrer à la version 3.x tout de suite ? Eh bien, il y en a quelques uns. La plupart des bibliothèques de modules que nous avons utlisées ne sont pas

```
> 2to3 pprint1v3.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored pprint1v3.py
--- pprint1v3.py (original)
+++ pprint1v3.py (refactored)
@@ -15,9 +15,9 @@
 objet1 = 3.00
 objet2 = 15.00
 # maintenant on affiche tout...
-print HautOuBas('=',40)
-print Fmt('Objet 1',30,objet1,10)
-print Fmt('Objet 2',30,objet2,10)
-print HautOuBas('-',40)
-print Fmt('Total',30,objet1+objet2,10)
-print HautOuBas('=',40)
+print(HautOuBas('=',40))
+print(Fmt('Objet 1',30,objet1,10))
+print(Fmt('Objet 2',30,objet2,10))
+print(HautOuBas('-',40))
+print(Fmt('Total',30,objet1+objet2,10))
+print(HautOuBas('=',40))
RefactoringTool: Files that need to be modified:
RefactoringTool: pprint1v3.py
```

```
> 2to3 -w pprint1v3.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored pprint1v3.py
--- pprint1v3.py (original)
+++ pprint1v3.py (refactored)
@@ -15,9 +15,9 @@
  objet1 = 3.00
  objet2 = 15.00
  # maintenant on affiche tout...
-print HautOuBas('=',40)
-print Fmt('Objet 1',30,objet1,10)
-print Fmt('Objet 2',30,objet2,10)
-print HautOuBas('-',40)
-print Fmt('Total',30,objet1+objet2,10)
-print HautOuBas('=',40)
+print(HautOuBas('=',40))
+print(Fmt('Objet 1',30,objet1,10))
+print(Fmt('Objet 2',30,objet2,10))
+print(HautOuBas('-',40))
+print(Fmt('Total',30,objet1+objet2,10))
+print(HautOuBas('=',40))
RefactoringTool: Files that were modified:
RefactoringTool: pprint1v3.py
```

disponibles en version 3.x pour le moment. Des choses comme Mutagen, que nous avons utilisé dans un article précédent, ne sont pas encore disponibles. Bien que ce soit une pierre d'achoppement, cela ne veut pas dire qu'il faut complètement abandonner Python 3.x.

Je vous suggère de commencer à coder en utilisant la syntaxe 3.x dès maintenant. Python 2.6 supporte presque tout ce dont vous pourriez avoir besoin d'écrire à la façon 3.x. Ainsi, vous serez prêt à passer à la version 3.x le jour où vous devrez le faire. Si vous pouvez vous contenter des bibliothèques de modules standards, allez-y et faites le changement. Par contre, si vous utilisez d'autres modules, il vous faudra attendre que la bibliothèque correspondante sorte en version 3.x. Cela viendra.

Voici quelques liens qui m'ont semblé utiles. Le premier est la page de manuel de 2to3. Le deuxième pointe vers un document de 4 pages d'astuces qui m'ont semblé une bonne référence. Le troisième est ce que je considère comme le meilleur livre pour utiliser Python (enfin jusqu'à ce que j'arrive à écrire le mien).

**À la prochaine.**

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

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

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

**Liens**

Manuel de 2to3 :
http://docs.python.org/library/2to3.html

*Passer de Python 2 à Python 3* (4 pages d'astuces) :
http://ptgmedia.pearsoncmg.com/imprint_downloads/informit/promotions/python/python2python3.pdf

*Plongez dans Python 3* :
http://diveintopython3.org/