Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume huit

Parties 44 à 48

Au sujet du Full Circle

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

Clause de non-responsabilité :

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

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

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

En réponse aux requêtes des lecteurs, nous avons réuni le contenu de certains articles consacrés à la programmation en python.

Pour l'instant, il s'agit d'une réédition directe de la série **Programmer en Python**, **parties 44 à 48**, des numéros 73 à 78, par l'incomparable professeur en Python Greg Walters.

Veuillez considérer l'origine de la publication ; les versions actuelles du matériel et des logiciels peuvent différer de ceux que nous présentons, ainsi vérifiez bien votre matériel et la version de vos logiciels avant d'émuler les tutoriels de cette édition spéciale. Vous pouvez installer des versions de logiciels plus récentes ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

SiteWeb :
http://www.fullcirclemagazine.org/

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

IRC : #fullcirclemagazine on chat.freenode.net

Équipe éditoriale :
Rédacteur en chef : Ronnie Tucker (pseudo : RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia (pseudo : admin / linuxgeekery-
admin@fullcirclemagazine.org

Nos remerciements vont à Canonical ainsi qu'aux nombreuses équipes de traduction à travers le monde.

TUTORIEL

Programmer en Python - Partie 44

Greg Walters

Nous allons laisser de côté ce mois-ci notre programme TVRage pour répondre en partie à une question d'un lecteur. On m'a demandé de parler de Qt Creator et comment l'utiliser pour concevoir des interfaces utilisateur pour des programmes Python.

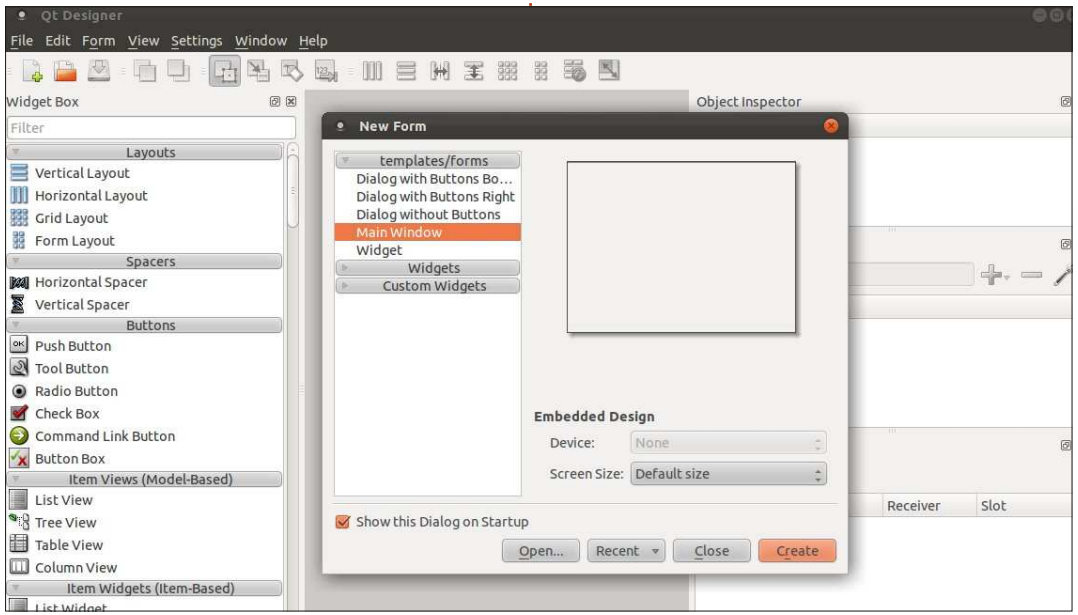
Malheureusement, que je sache, le support de Qt Creator n'est pas encore prêt pour Python. Il est effectivement en cours d'élaboration, mais n'est pas tout à fait « prêt pour les heures de grande écoute ».

Ainsi, afin de nous préparer pour ce futur article, nous allons travailler avec QT4 Designer. Vous aurez besoin d'installer (si ce n'est pas déjà fait) python-qt4, qt4-dev-tools, python-qt4-dev, pyqt4-dev-tools et libqt4-dev.

Une fois que c'est fait, vous trouverez QT4 Designer sous Applications > Programmation. Allez-y et lancez-le. Vous devriez voir quelque chose comme ceci :

Assurez-vous que « Main Window » est sélectionné et cliquez sur le bouton « Créer ». Vous verrez alors un formulaire vierge sur lequel vous pouvez faire glisser et déplacer des contrôles.

La première chose à faire est de redimensionner la fenêtre principale. Réglez-la à environ 500×300. Vous pouvez voir sa taille en regardant l'éditeur de propriétés dans la partie « géométrie » sur le côté droit de la fenêtre du concepteur. Maintenant, faites défiler vers le bas la liste de l'éditeur de propriétés jusqu'à ce que vous voyiez « windowTitle ». Remplacez le texte « MainWindow » par « Python Test1 ». Vous devriez voir le changement dans la barre de titre de la fenêtre de conception : « Python Test1 - untitled* ». C'est maintenant un bon moment pour sauvegarder notre projet. Nommez-le « pytest1.ui ». Ensuite, nous allons mettre un bouton sur notre formulaire. Ce sera un bouton pour quitter le programme de test. Sur le côté gauche de la fenêtre du concepteur, vous verrez toutes les commandes qui sont disponibles. Trouvez la section « Buttons » et faites glisser un « Push Button » sur le formulaire. Contrairement aux concepteurs graphiques que nous avons utilisés dans le passé, vous n'avez pas à créer des grilles pour contenir vos contrôles lorsque vous utilisez QT4 Designer. Déplacez le bouton presque au centre en bas du formulaire. Si vous regardez l'éditeur de propriétés sous « géométrie », vous verrez quelque chose comme ceci :




```
[(200,260), 97x27]
```

Dans les parenthèses se trouvent les positions X et Y de l'objet (bouton-poussoir dans ce cas) sur le formulaire, suivies par sa largeur et sa hauteur. J'ai positionné le mien en 200,260.

Juste au-dessus se trouve la propriété « objectName » qui, par défaut, est réglée sur « pushButton ». Changez cela en « btnQuitter». Maintenant, faites défiler vers le bas la liste de l'éditeur de propriétés jusqu'à la section « QAbstractButton », et définissez la propriété « text » à « Quitter ». Vous pouvez voir sur notre formulaire que le texte sur le bouton a changé.

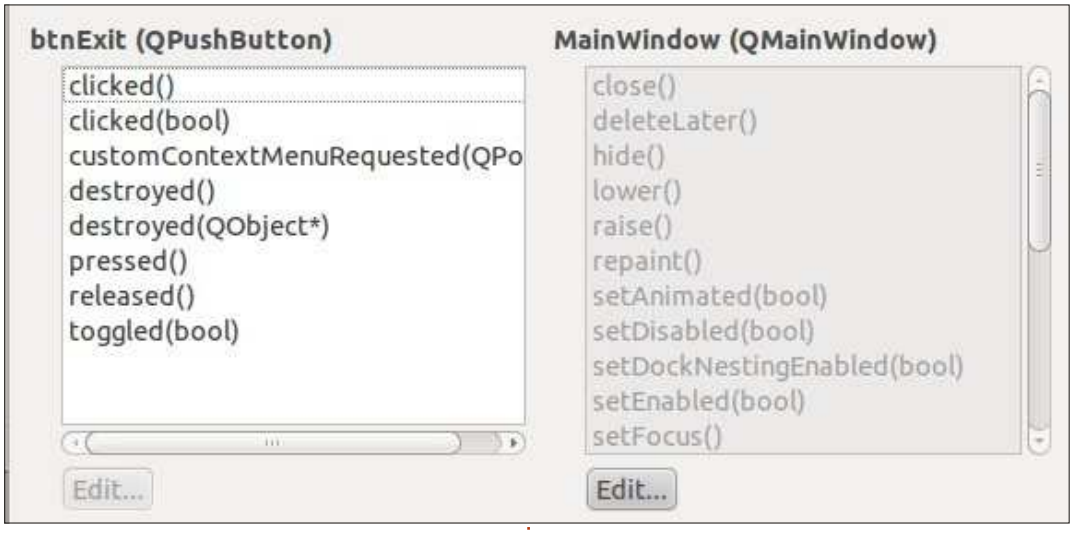
Maintenant, ajoutez un autre bouton et positionnez-le en 200,200. Réglez sa propriété objectName à « btnCliqueMoi » et son texte à « Cliquez-moi ! ».

Ensuite, ajoutez une étiquette (Label). Vous la trouverez dans la boîte à outils sur la gauche, sous « Display Widgets ».

Mettez-la à proximité du centre du formulaire (j'ai mis la mienne en 210,130), et réglez sa propriété objectName à lblAffichage. Nous voulons la rendre plus grande qu'elle n'est par défaut, réglez donc sa taille à quelque chose comme 221 x 20. Dans l'éditeur de propriétés, descendez jusqu'à la section « QLabel », et définissez l'alignement horizontal à « AlignHCenter » (AligneementCentreH). Mettez le texte à vide. Nous réglerons le texte dans le code lorsque le btnCliqueMoi sera cliqué. Maintenant, sauvegardez à nouveau le projet.

Slots et signaux

La section suivante est peut-être un peu difficile à faire entrer dans votre tête, surtout si vous avez été avec nous pendant une longue période et avez utilisé les concepteurs graphiques précédents. Dans les autres concepteurs, nous avons utilisé des événements qui se sont déclenchés quand on a cliqué sur un objet, comme un bouton. Avec QT4 Designer, les événements s'ap-pellent des signaux et la fonction appelée par ce signal s'appelle un slot. Donc, pour notre bouton Quitter, nous utilisons le signal « cliquer » pour appeler le slot « fermeture de la fenêtre principale ». Êtes-vous totalement perdus maintenant ? Je l'étais quand



j'ai eu affaire à QT au début, mais cela commence à sembler logique après un certain temps.

Heureusement, il existe un moyen très simple pour utiliser les slots et les signaux prédéfinis. Si vous appuyez sur la touche F4 du clavier, vous irez dans le mode d'édition des signaux et des slots. (Pour sortir du mode d'édition, appuyez sur F3.) Maintenant, faites un clic gauche et maintenez le bouton Quitter enfoncé, puis faites glisser légèrement vers le haut et vers la droite, en dehors du bouton et sur le formulaire principal, puis relâchez le clic. Vous verrez une fenêtre de dialogue qui ressemble à celle ci-dessus.

Cela nous donnera un moyen facile de connecter le signal « clicked » au formulaire. Sélectionnez la première option sur la gauche qui devrait être « clicked() ». Cela activera la partie droite de la fenêtre et sélectionnera l'option « close() » dans la liste, puis cliquez sur « OK ». Vous verrez quelque chose qui ressemble à ceci :

Le signal de clic (événement) est lié à la routine de clôture de la fenêtre principale.

Pour le signal clicked de btnCliqueMoi, nous le gérerons dans le code.

Enregistrez le fichier une fois de plus. Quittez QT4 Designer et ouvrez un terminal. Allez dans le répertoire où vous avez enregistré le fichier. Maintenant, nous allons générer un fichier python en utilisant l'outil en ligne de commande pyuic4. Cela va lire le fichier .ui. La commande sera :

```
pyuic4 -x pytest1.ui -o
pytest1.py
```

Le paramètre -x indique d'inclure le code pour exécuter et afficher l'interface utilisateur. Le paramètre -o indique de créer un fichier de sortie plutôt que de simplement afficher le fichier sur la sortie standard. Une chose importante à noter ici : assurez-vous d'avoir vraiment tout fait dans QT4 Designer avant de créer le fichier python, sinon il sera complètement réécrit et vous devrez recommencer à partir de zéro.

Une fois que vous aurez fait cela, vous aurez votre fichier python. Ouvrez-le dans votre éditeur de texte favori.

Le fichier lui-même ne contient que 60 lignes environ, y compris les commentaires. Nous n'avons placé que quelques contrôles, c'est pour cela qu'il n'est pas très long. Je ne vais pas montrer beaucoup de code. Vous devriez être en mesure de suivre la plupart du code maintenant. Cependant, nous allons créer et ajouter d'autres codes afin de mettre la

fonctionnalité pour définir le texte de l'étiquette.

La première chose que nous devons faire est de copier la ligne de signal et de slot et la modifier. Quelque part autour de la ligne 44 devrait se trouver le code suivant :

```
QtCore.QObject.connect(self.b
tnQuitter,
QtCore.SIGNAL(_fromUtf8("clic
ked()")), MainWindow.close)
```

Copiez-le, et recollez-le juste en dessous. Puis modifiez-le en :

```
QtCore.QObject.connect(self.b
tnCliqueMoi,
QtCore.SIGNAL(_fromUtf8("clic
ked()")),
self.reglerAffichage)
```

Ceci va créer la connexion signal/slot pour notre routine qui va régler le texte de l'étiquette. Sous la routine retranslateUi, ajoutez le code suivant :

```
def reglerAffichage(self):

self.lblAffichage.setText(_fr
omUtf8("Ca chatouille!!!"))
```

J'ai trouvé le nom setText dans la ligne d'initialisation dans la routine setupUi.

Maintenant, exécutez votre code. Tout devrait fonctionner comme prévu.

Bien que ce soit un exemple très simple, je suis sûr que vous êtes assez avancé pour jouer avec QT4 Designer et vous faire une idée de la puissance de l'outil.

Le mois prochain, nous reviendrons de notre détour et commencerons à travailler sur l'interface utilisateur pour notre programme de TVRage.

Comme toujours, le code peut être trouvé sur pastebin à http://pastebin.com/FniB3s85 pour le code .ui et http://pastebin.com/K7zViFu3 pour le code python. [NdT: code traduit par l'équipe francophone.]

Pour le code original, voir http://pastebin.com/98fSasdb pour le code .ui et http://pastebin.com/yC30B885 pour le code python.

Rendez-vous la prochaine fois.

MON HISTOIRE COURTE

par Anthony Venable

Cette histoire commence début 2010. J'étais sans le sou à ce moment et j'essayais de trouver un système d'exploitation gratuit. J'avais besoin d'un qui puisse tourner sur mes PC à la maison. J'avais cherché sur internet mais n'ai rien trouvé pendant longtemps. Un jour, à Barnes & Noble [Ndt : grande librairie], j'ai vu un magazine sur Linux. (Bien que j'en aie entendu parler avant, je n'y avais jamais pensé comme à quelque chose que je serais capable d'utiliser.) Quand j'ai demandé à des personnes que je savais être des professionnels de l'informatique, ils m'ont répondu que c'était pour les experts et que c'était difficile à utiliser. Je n'entendais jamais rien de positif sur Linux. Je suis vraiment étonné de ne pas être tombé dessus plus tôt.

Quand j'ai lu le magazine, j'ai découvert Ubuntu 9.10 - Karmic Koala. Cela avait l'air vraiment très bien, comme si c'était exactement ce que je recherchais. Par conséquent, très excité, je l'ai ramené à la maison et, à ma grande surprise, il fut tellement facile de l'installer sur mon PC que j'ai décidé de le laisser en parallèle avec Windows XP en double amorçage. Je n'ai fait que mettre le CD dans le lecteur et les instructions étaient si détaillées qu'il aurait fallu être très limité pour ne pas comprendre comment configurer le tout.

Depuis, je suis très satisfait d'Ubuntu en général et j'ai pu tester les versions ultérieures comme la 10.04 (Maverick Meerkat) et la 10.10 (Lucid Lynx). J'attends les futures versions pour voir comment ils intégreront le contrôle multitactile encore mieux que dans la 10.04.

Cette expérience montre, encore une fois, comment j'arrive à trouver les meilleurs trucs par accident.

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

Programmer en Python - Partie 45

Greg Walters

Cette fois-ci, nous allons retravailler notre programme de base de données à partir des quelques articles précédents (les parties 41, 42 et 43 dans les numéros 70, 71 et 72). Puis, au cours des prochains articles, nous allons utiliser QT pour créer l'interface utilisateur.

Tout d'abord, regardons comment fonctionne l'application existante. Voici un aperçu brut :

- Créer une connexion à la base de données, qui crée la base de données si nécessaire.
- Créer un pointeur sur la base de données.
- Créer la table si elle n'existe pas.
- Attribuer le(s) dossier(s) vidéo à une variable.
- Rechercher les fichiers vidéo dans le(s) dossier(s)
- Obtenir le nom du fichier, le nom de la série, le numéro de la saison, le numéro de l'épisode.
- Vérifiez si l'épisode existe dans la base de données.
- S'il n'y est pas, l'ajouter à la base de données avec un « -1 » comme ID TvRage.
- Parcourir ensuite la base de données, obtenir l'id de la série et le statut, si nécessaire, et mettre à jour la base de données.

Nous allons repenser la base de données pour inclure une autre table et modifier la table de données existante. Tout d'abord, nous allons créer notre nouvelle table appelée Series. Elle contiendra toutes les informations sur les séries TV que nous avons sur notre système. La nouvelle table comprendra les champs suivants :

- PKID.
- Nom de la série.
- ID série TvRage.
- Nombre de saisons.
- Date de début.
- Drapeau terminée.
- Pays d'origine.
- Etat de la série (terminé, courant, etc.).
- Classification (à partir d'un script, « réalité », etc.).
- Résumé de l'intrigue de la série.
- Genre.
- Durée en minutes.
- Chaîne de diffusion.
- Jour de diffusion dans la semaine.
- Horaire de diffusion.
- Chemin de la série.

Nous pouvons utiliser la routine FabriquerBase existante pour créer notre nouvelle table. Avant le code existant, ajoutez le code ci-dessus.

```
sql = 'CREATE TABLE IF NOT EXISTS Series (
        pkid INTEGER PRIMARY KEY AUTOINCREMENT,
        NomSerie TEXT,
        SerieID TEXT,
        Saison TEXT,
        DateDebut TEXT,
        Terminee TEXT,
        PaysOrigine TEXT,
        Etat TEXT,
        Classification TEXT,
        Resume TEXT,
        Genre TEXT,
        Duree TEXT,
        Reseau TEXT,
        JourDiffusion TEXT,
        HoraireDiffusion TEXT,
        Chemin TEXT);'
cursor.execute(sql)
```

L'instruction SQL (“sql = …”) doit être sur une seule ligne, mais est éclatée ici pour faciliter votre compréhension. Nous laisserons la modification de la table existante pour plus tard.

Maintenant, nous devons modifier notre routine ParcourirChemin pour enregistrer le nom de la série et le chemin dans la table de Series.

Remplacez la ligne qui dit

```
sqlquery = 'SELECT
count(pkid) as rowcount from
TvShows where NomFichier =
"%s";' % fl
```

par

```
sqlquery = 'SELECT
count(pkid) as rowcount from
Series where NomSerie =
"%s";' % NomEmission
```

Cela (pour vous rafraîchir la mémoire) va vérifier si nous avons déjà mis les séries dans la table.

Maintenant, trouver les deux lignes qui disent :

```
 sql = 'INSERT INTO
EmissionsTV
(Series,CheminRacine,NomFichi
er,saison,episode,tvrageid)
VALUES (?,?,?,?,?,?)'

cursor.execute(sql,(NomEmissi
on,Racine,fl,saison,episode,-
1))
```

et les remplacer par :

```
sql = 'INSERT INTO Series
(NomSerie,Chemin,SerieID)
VALUES (?,?,?)'

cursor.execute(sql,(NomEmissi
on,Racine,-1))
```

Ceci va insérer le nom de la série (NomEmission), le chemin de la série, et un « -1 » comme identifiant de TvRage. Nous utilisons le « -1 » comme un drapeau pour savoir que nous avons besoin d'obtenir l'information de série de TvRage.

Ensuite, nous allons retravailler la routine ParcourirBase pour alimenter ces séries pour lesquelles nous n'avons pas d'informations (SeriesID = -1) et mettre à jour ce dossier.

Modifiez la chaîne de requête de :

```
sqlstring = "SELECT DISTINCT
series FROM EmissionsTV WHERE
tvrageid = -1"
```

en

```
sqlstring = "SELECT
pkid,NomSerie FROM Series
WHERE SerieID = -1"
```

Cela va créer un ensemble de résultats que nous pourrons ensuite utiliser pour interroger TvRage pour chaque série. Maintenant, trouver/remplacer les deux lignes suivantes :

```
 NomSerie = x[0]

searchname =
string.capwords(x[0]," ")
```

par

```
pkid = x[0]

NomSerie = x[1]

searchname =
string.capwords(x[1]," ")
```

Nous allons utiliser le PKID pour la déclaration de mise à jour. Ensuite, nous devrons modifier l'appel à la routine MettreAJourBase pour inclure le PKID.

Modifiez la ligne :

```
MettreAJourBase(NomSerie,id)
```

en

```
MettreAJourBase(NomSerie,id,p
kid)
```

et changez la ligne :

```
RecupererEtatEmission(NomSeri
e,id)
```

en

```
RecupererDonneesEmission(NomS
erie,id,pkid)
```

```
def RecupererDonneesEmission(NomSerie,id,pkid):
    tr = TvRage()
    idcursor = connection.cursor()
    dict = tr.TrouveInfoEmission(id)
```

```
Saison = dict['Saison']
    DateDebut = dict['Date Debut']
    Terminee = dict['Terminee']
    PaysOrigine = dict['PaysOrigine']
    Etat = dict['Etat']
    Classification = dict['Classification']
    Resume = dict['Resume']
```

qui sera une nouvelle routine ; nous allons la créer dans un instant.

Ensuite, modifiez la définition de la routine de MettreAJourBase de :

```
def
MettreAJourBase(NomSerie,id):
```

à :

```
def
MettreAJourBase(NomSerie,id,P
KID):
```

Ensuite, nous devons changer la chaîne de requête de :

```
sqlstring = 'UPDATE
EmissionsTV SET tvrageid = '
+ id + ' WHERE Series = "' +
NomSerie + '"'
```

en :

```
sqlstring = 'UPDATE Series
SET SerieID = ' + id + '
WHERE pkID = %d' % pkid
```

Maintenant, nous devons créer la routine RecupererDonneesEmission (ci-dessus). Nous allons obtenir les informations de TvRage et les insérer dans la table Series.

En guise d'aide-mémoire : nous créons une instance des routines de TvRage et un dictionnaire qui contient les informations sur notre série. Nous allons ensuite créer des variables pour contenir les données de mise à jour du tableau (ci-dessus).

Rappelez-vous que Genres vient en tant que sous-élément et contient une ou plusieurs listes de genres. Heureusement, quand nous avons codé les routines de TvRage, nous avons créé

une chaîne qui contient tous les genres, peu importe combien sont retournés ; nous pouvons ainsi tout simplement utiliser la chaîne de caractères genre :

```
genre = dict['Genre']
duree = dict['Duree']

reseau = dict['Reseau']

jourdiffusion =
dict['JourDiffusion']
horairediffusion
= dict['HoraireDiffusion']
```

Enfin, nous créons la chaîne de requête pour réaliser la mise à jour (en bas). Encore une fois, tout cela doit être sur une seule ligne, mais je l'ai cassée ici pour la rendre facile à comprendre.

La partie {chiffre} (à titre d'information) est similaire à l'option de formatage « %s ». Cela crée notre chaîne de requête en remplaçant le {chiffre} avec les données réelles que nous voulons. Puisque nous avons déjà défini tous ces champs comme texte, nous devons utiliser les guillemets doubles (") pour encadrer les données ajoutées.

Et enfin, nous écrivons à la base de données (ci-dessous).

C'est tout pour cette fois-ci. La prochaine fois, nous allons continuer comme j'ai expliqué au début de l'article. Jusqu'à la prochaine fois, **amusez-vous bien !**

```
    try:
        idcursor.execute(sqlstring)
    except:
        print "Une erreur est survenue dans l'ajout des
informations de la série."
```

```
sqlstring = 'Update Series SET Saison = "{0}", DateDebut = "{1}", Terminee = "{2}",
PaysOrigine = "{3}", Etat = "{4}", Classification = "{5}",
Resume = "{6}", Genre = "{7}", Duree = "{8}", Reseau = "{9}",
JourDiffusion = "{10}", HoraireDiffusion = "{11}"
WHERE pkID ={12}'.format(Saison,DateDebut,Terminee,
PaysOrigine,Etat,Classification,Resume,
Genre,Duree,Reseau,JourDiffusion,HoraireDiffusion,pkid)
```

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Le Podcast Ubuntu couvre toutes les dernières nouvelles et les problèmes auxquels sont confrontés les utilisateurs de Linux Ubuntu et les fans du logiciel libre en général. La séance s'adresse aussi bien au nouvel utilisateur qu'au plus ancien codeur. Nos discussions portent sur le développement d'Ubuntu, mais ne sont pas trop techniques. Nous avons la chance d'avoir quelques supers invités, qui viennent nous parler directement des derniers développements passionnants sur lesquels ils travaillent, de telle façon que nous pouvons tous comprendre ! Nous parlons aussi de la communauté Ubuntu et de son actualité.

Le podcast est présenté par des membres de la communauté Ubuntu Linux du Royaume-Uni. Il est couvert par le Code de Conduite Ubuntu et est donc adapté à tous.

L'émission est diffusée en direct un mardi soir sur deux (heure anglaise) et est disponible au téléchargement le jour suivant.

podcast.ubuntu-uk.org

TUTORIEL

Programmer en Python - Partie 46

Greg Walters

Habituellement, mes articles sont assez longs. Toutefois, en raison de certains problèmes médicaux, celui de ce mois-ci sera assez court (dans le grand ordre des choses). Cependant, nous allons pouvoir continuer et approfondir notre série sur le programme de gestion de données.

Une des choses que notre programme fera pour nous est de nous avertir si nous avons des épisodes manquants dans une des séries de la base de données. Voici le scénario. Nous avons une série, appelons-la « La série des années 80 », qui a duré trois saisons. Dans la saison 2, il y avait 15 épisodes. Cependant, nous n'en avons que 13 dans notre bibliothèque. Comment trouver quels épisodes manquent – informatiquement ?

Le plus simple est d'utiliser des listes et des ensembles. Nous avons déjà utilisé des listes dans un certain nombre d'articles au cours des quatre dernières années, mais les ensembles sont un nouveau type de données dans cette série ; nous allons donc les examiner pendant quelque temps. Selon la « documentation officielle » pour Python (docs.python.org), voici la définition d'un ensemble (« set ») :

« Un ensemble est une collection sans notion d'ordre et sans doublon. Les utilisations de base comprennent des tests d'appartenance et l'élimination des doublons. On peut également faire des opérations mathématiques sur les ensembles comme l'union, l'intersection, la différence et la différence symétrique. »

Je vais continuer à utiliser l'exemple de la page de documentation pour illustrer le processus :

```
>>> panier = ['pomme',
'orange', 'pomme', 'poire',
'orange', 'banane']

>>> fruit = set(panier)

>>> fruit

set(['orange', 'poire',
'pomme', 'banane'])
```

Remarquez que dans la liste originale qui a été affectée à la variable panier, les éléments pomme et orange ont été mis en double, mais quand nous l'avons affectée à un ensemble, les doublons ont été éliminés. Maintenant, pour utiliser l'ensemble que nous venons de créer, nous pouvons vérifier si un fruit (ou autre chose) est dans l'ensemble. Nous pouvons utiliser l'opérateur « dans » (« in ») :

```
>>> 'orange' in fruits

vrai

 >>> 'kiwi' in fruits

faux

>>>
```

C'est assez simple et, je l'espère, vous commencez à comprendre où je veux en venir. Disons que nous avons une liste de courses qui contient plein de fruits et, en parcourant le magasin, nous voulons vérifier ce qui nous manque – c'est-à-dire les éléments de la liste de courses qui ne sont pas dans notre panier. Nous pouvons commencer comme ceci :

```
>>> listecourses = ['orange',
'pomme', 'poire', 'banane',
'kiwi', 'raisin']

>>> panier = ['pomme',
'kiwi', 'banane']

>>> lc = set (listecourses)

>>> p = set (panier)

>>> lc-p

set(['orange', 'poire',
'raisin'])

>>>
```

Nous créons nos deux listes, listecourses pour ce que nous voulons et panier pour ce que nous avons. Nous affectons chacune à un ensemble et utilisons l'opérateur d'ensemble « différence » (le signe moins) pour nous donner les éléments qui sont dans la liste des courses, mais pas dans le panier.

Maintenant, en utilisant la même logique, nous allons créer une routine (page suivante, en bas à gauche) qui traitera de nos épisodes manquants. Nous allons appeler notre routine « RechercherManquants » et lui passer deux variables. La première est un entier réglé au nombre d'épisodes de cette saison et le second est une liste contenant les numéros d'épisodes que nous avons pour cette saison.

La routine, lorsque vous l'exécutez, affiche [5, 8, 15] ce qui est cor-

rect. Maintenant, regardons le code. La première ligne crée un ensemble appelé EpisodesNecessaires qui est une liste de nombres entiers créée à l'aide de la fonction de plage de nombres. Nous devons donner à la fonction de plage la valeur de début et la valeur de fin. Nous ajoutons 1 à la valeur haute pour nous donner la liste correcte des valeurs de 1 à 15. Rappelez-vous que la fonction de plage part en fait de 0, donc quand nous lui donnons 16 (le 15 prévu + 1), la liste réellement créée va de 0 à 15. Nous disons à la fonction de plage de commencer à 1, puisque même si la plage est de 0 à 15, soit 16 valeurs, nous en voulons 15 à partir de 1.

Ensuite, nous créons un ensemble avec la liste qui est passée à notre routine, qui contient les numéros d'épisodes que nous avons déjà.

Maintenant, nous pouvons créer une liste en utilisant l'opérateur de différence d'ensemble sur les deux ensembles. Nous faisons cela pour pouvoir le trier avec la méthode list.sort(). Vous pouvez certainement renvoyer la liste si vous le souhaitez, mais dans cette itération de la routine, nous allons simplement l'afficher.

Eh bien, me voici au bout du temps que mon corps peut supporter assis dans le fauteuil en face de l'ordinateur, donc je vais vous laisser pour ce mois-ci avec la question de comment nous allons utiliser tout ceci dans notre gestionnaire de médias.

Passez un bon mois et à bientôt.

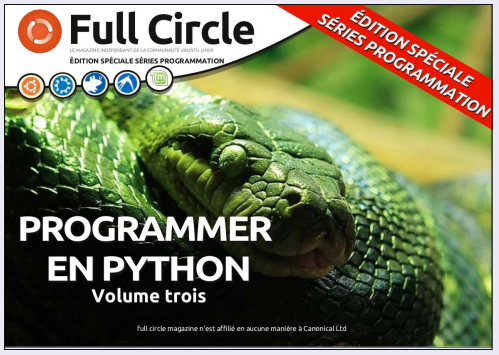
```
def RechercherManquants(attendus,dejapresents):
#==================================
# attendus : numeros de tous les episodes requis
# dejapresents : liste des episodes que nous avons deja
# renvoie une liste triee des numeros manquants
#==================================
EpisodesRequis = set(range(1,attendus+1))
EpisodesPresents = set(dejapresents)
EncoreBesoin = list(EpisodesRequis - EpisodesPresents)
EncoreBesoin.sort()
print EncoreBesoin

RechercherManquants(15,[1,2,3,4,6,7,9,10,11,12,13,14])
```

Éditions Spéciales Python :

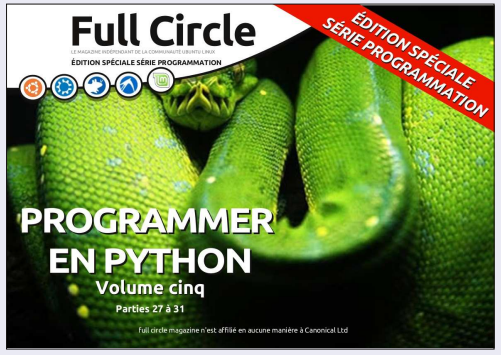
http://www.fullcirclemag.fr/?download/224

http://www.fullcirclemag.fr?download/230



www.fullcirclemag.fr/?download/231

http://www.fullcirclemag.fr/?download/240



http://www.fullcirclemag.fr/?download/268

http://www.fullcirclemag.fr/?download/272

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

Programmer en Python - Partie 47

Greg Walters

Le mois dernier, nous avons parlé de l'utilisation des ensembles pour afficher les épisodes manquants. Voici venu le moment de mettre en pratique le code brut que nous avons présenté.

Nous allons modifier une routine et en écrire une. Nous ferons d'abord la modification. Dans le fichier de travail que vous avez utilisé ces derniers mois, cherchez la routine ParcourirChemin(chemin). Les quatrième et cinquième lignes devraient être :

```
ficerr =
open('erreurs.log',"w")

for racine, reps, fichiers in
os.walk(chemin,topdown=True):
```

Entre ces deux lignes, nous allons insérer le code suivant :

```
derniere_racine = ''

ep_liste = []

emission_courante = ''

saison_courante = ''
```

À présent, vous devriez reconnaître que tout ce que nous faisons ici est l'initialisation de variables. Il y a trois variables de chaîne et une liste. Nous allons utiliser la liste pour contenir les numéros d'épisodes (d'où le nom ep_liste).

```
for racine, reps, fichiers in os.walk(chemin,topdown=True):
    for fic in [f for f in fichiers if f.endswith (('.avi','mkv','mp4','m4v'))]:
```

Jetons un coup d'œil rapide et rafraîchissons notre mémoire (ci-dessus) sur ce que nous faisons dans la routine existante avant de la modifier.

Les deux premières lignes ici initialisent les choses pour la routine ParcourirChemin, dans laquelle nous commençons dans un dossier donné dans le système de fichiers et visitons de manière récursive chaque dossier en-dessous et vérifions l'existence des fichiers qui ont l'extension .avi, .mkv, .mp4 ou .m4v. S'il y en a, nous parcourons ensuite la liste de ces noms de fichiers.

Dans la ligne en haut à droite, nous appelons la routine RecupereSaisonEpisode pour récupérer le nom de la série et les numéros de la saison et de l'épisode à partir du nom de fichier. Si tout est analysé correctement, la variable estok est réglée à vrai (true) et les données que nous cherchons sont placées dans une liste qui nous est retournée.

```
 # Combine chemin et nom de fichier pour creer une seule variable
fn = join(racine,fic)
NomFicOriginal,ext = os.path.splitext(fic)
fl = fic
estok,donnees = RecupereSaisonEpisode(fl)
```

Ensuite (ci-dessous), nous affectons simplement les données retournées par RecupereSaisonEpisode et les mettons dans des variables distinctes avec lesquelles nous pourrons jouer. Maintenant que nous savons où nous en sommes, parlons de ce qui va se passer.

Nous voulons obtenir le numéro d'épisode pour chaque fichier et le placer dans la liste ep_liste. Une fois que nous avons terminé avec tous les fichiers du dossier dans lequel nous sommes, nous pouvons faire l'hypothèse que nous avons à peu près correctement traité les fichiers et que le numéro d'épisode le plus élevé est le plus récent disponible. Comme nous l'avons vu le mois dernier, nous pouvons alors créer un ensemble contenant les numéros de 1 au dernier épisode, convertir la liste en ensemble et calculer la différence. C'est très bien en théorie, mais il y a un petit grain de sable dans notre mécanique quand on la met en pratique. Nous n'avons pas vraiment d'indication claire et nette quand nous avons fini avec un dossier particulier. Ce que nous savons cependant, c'est que, lorsque nous aurons fini avec chaque fichier, le code juste

```
if estok:
    nomEmission = data[0]
    saison = data[1]
    episode = data[2]
    print("Saison {0} Episode {1}".format(saison,episode)
```

après « for fic in [… » est exécuté. Si nous connaissons le nom du dernier dossier visité et le nom du dossier en cours, nous pouvons comparer les deux et s'ils sont différents, c'est que nous venons de terminer un dossier et que notre liste d'épisodes devrait être complète. C'est pour cela que la variable « derniere_racine » est là.

Nous allons mettre la majorité de notre nouveau code juste après la ligne « for fic in […. Cela ne représente que sept lignes, que voici (j'ai mis en noir les lignes existantes pour plus de clarté).

```
for fic in [f for f in fichiers if f.endswith (('.avi','mkv','mp4','m4v'))]:
    # Combine chemin et nom de fichier pour creer une seule variable
    if derniere_racine != racine:
      derniere_racine = racine
      if len(ep_liste) > 0:
          Manquants(ep_liste,max(ep_liste),saison_courante,emission_courante)
      ep_liste = []
      emission_courante = ''
      saison_courante = ''
    fn = join(racine,fic)
```

Voici la logique du nouveau code, ligne par ligne :

Tout d'abord, nous vérifions si la variable derniere_racine a la même valeur que racine (le nom du dossier en cours). Si c'est le cas, nous sommes dans le même dossier et nous n'exécutons pas le code. Sinon, nous attribuons le nom du dossier actuel à la variable derniere_racine. Ensuite, nous vérifions si la liste des épisodes (ep_liste) a des entrées (len(ep_liste)> 0). C'est pour s'assurer que nous ne sommes pas dans un répertoire vide. Si nous avons des éléments dans la liste, alors nous appelons la routine Manquant. Nous lui passons la liste des épisodes, le numéro de l'épisode le plus élevé, le numéro de la saison en cours, et le nom de la saison, pour pouvoir les afficher plus tard. Les trois dernières lignes effacent la liste, le nom de l'émission en cours, et la saison en cours, et nous continuons comme nous le faisions avant.

Ensuite, nous devons changer deux lignes et ajouter une ligne de code dans le morceau de code « if estok » quelques lignes plus bas. Encore une fois, les lignes noires sont le code existant :

```
        estok,donnees = RecupereSaisonEpisode(fl)
                if estok:
                emission_courante = nomEmission = data[0]
                saison_courante = saison = data[1]
                episode = data[2]
                ep_liste.append(int(episode))
        else:
```

Ici, nous venons de revenir de la routine RecupereSaisonEpisode. Si nous avions un nom de fichier analysable, nous voulons obtenir le nom de l'émission et le numéro de la saison, et ajouter l'épisode actuel dans la liste. Remarquez que nous convertissons en nombre entier le numéro de l'épisode avant de l'ajouter à la liste.

Nous en avons fini avec cette partie du code. Maintenant, tout ce qui nous reste à faire est d'ajouter la routine Manquant. Juste après la routine ParcourirChemin, nous allons ajouter le code suivant.

Encore une fois, c'est du code très simple et nous l'avons quasiment déjà vu le mois dernier, mais nous allons le parcourir juste au cas où vous l'auriez manqué.

Nous définissons la fonction et mettons en place quatre paramètres. Nous passerons la liste des épisodes (episode_liste), le nombre d'épisodes que nous devrions avoir (nombre_theorique) qui est le numéro d'épisode le plus élevé dans la liste des épisodes, le numéro de la saison (saison), et le nom de l'émission (nom_emission).

```
#--------------------------------
def Manquants(episode_liste,nombre_theorique,saison,nom_emission):
    temp = set(range(1,nombre_theorique+1))
    ret = list(temp-set(episode_liste))
    if len(ret) > 0:
        print('Episodes manquants pour {0} saison {1} -
{2}'.format(nom_emission,saison,ret))
```

Ensuite, nous créons un ensemble qui contient une liste de numéros en utilisant la fonction intégrée « range », en allant de 1 à la valeur nombre_theorique + 1. Nous appelons ensuite la fonction de différence, sur cet ensemble et un ensemble converti depuis la liste des épisodes (temp - set(episode_liste)), et le reconvertissons en liste. Nous vérifions ensuite s'il y a quelque chose dans la liste, afin de ne pas afficher une ligne avec une liste vide, et s'il y a quelque chose, nous l'affichons.

C'est tout. La seule faille dans cette logique est qu'avec cette façon de faire, nous ne savons pas s'il y a de nouveaux épisodes que nous n'avons pas.

J'ai mis les deux routines sur pastebin au cas où vous voudriez juste faire un remplacement rapide dans votre code. Vous pouvez les consulter ici : http://pastebin.com/CnHK8xxf.

Passez un bon mois et nous vous reverrons bientôt.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Le Podcast Ubuntu couvre toutes les dernières nouvelles et les problèmes auxquels sont confrontés les utilisateurs de Linux Ubuntu et les fans du logiciel libre en général. La séance s'adresse aussi bien au nouvel utilisateur qu'au plus ancien codeur. Nos discussions portent sur le développement d'Ubuntu, mais ne sont pas trop techniques. Nous avons la chance d'avoir quelques supers invités, qui viennent nous parler directement des derniers développements passionnants sur lesquels ils travaillent, de telle façon que nous pouvons tous comprendre ! Nous parlons aussi de la communauté Ubuntu et de son actualité.

Le podcast est présenté par des membres de la communauté Ubuntu Linux du Royaume-Uni. Il est couvert par le Code de Conduite Ubuntu et est donc adapté à tous.

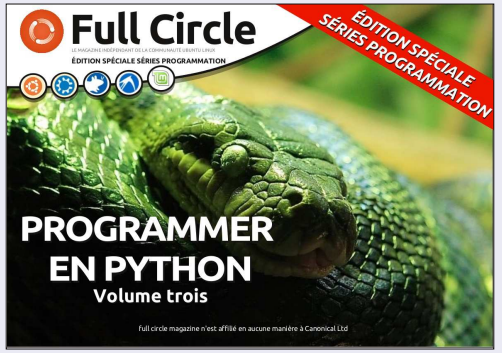
L'émission est diffusée en direct un mardi soir sur deux (heure anglaise) et est disponible au téléchargement le jour suivant.

podcast.ubuntu-uk.org

Éditions spéciales Python :

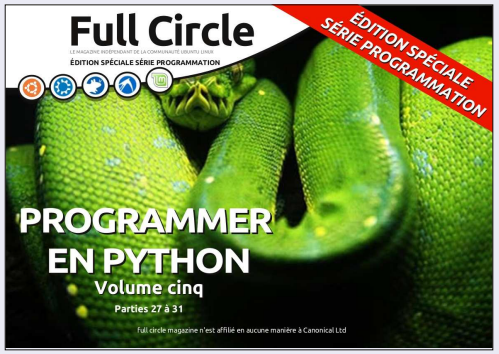
http://www.fullcirclemag.fr/?download/224

http://www.fullcirclemag.fr?download/230



http://www.fullcirclemag.fr/?download/231

http://www.fullcirclemag.fr/?download/240



http://www.fullcirclemag.fr/?download/268

http://www.fullcirclemag.fr/?download/272

Programmer en Python - Partie 48

Greg Walters

Bienvenue Il est difficile d'imaginer que ça fait 4 ans que j'ai commencé cette série. Je pensais laisser de côté le projet de gestionnaire de média un instant et revenir à certaines bases de la programmation Python.

Ce mois-ci, je vais revisiter la commande d'affichage (print). C'est l'une des fonctions les plus utilisées (au moins dans ma programmation) qui semble ne jamais obtenir le détail qu'elle mérite. Il y a beaucoup de choses que vous pouvez faire avec elle en dehors du basique '%s %d'. Puisque la syntaxe de la fonction print est différente entre Python 2.x et 3.x, nous allons les examiner séparément. Rappelez-vous, cependant, que vous pouvez utiliser la syntaxe 3.x dans Python 2.7. La plus grande part de tout ce que je vous présente ce mois-ci se fera à partir de la ligne de commande interactive. Vous pouvez suivre au fur et à mesure. Le code ressemblera à ceci :

```
>>> a = “Salut Python”
>>> print(“La chaine a est
%s” % a)
```

et la sortie sera en gras, comme ceci :

```
La chaine a est Salut Python
```

Python 2.x

Bien sûr, vous vous souvenez que la syntaxe simple pour la fonction print en 2.x utilise la substitution de variable de %s ou %d pour les chaînes simples ou les nombres décimaux. Mais beaucoup d'autres options de mise en forme sont disponibles. Par exemple, si vous avez besoin de formater un nombre avec des zéros à gauche, vous pouvez le faire de cette façon :

```
>>> print(“Votre valeur est
%03d” % 4) Votre valeur est
004
```

Dans ce cas, nous utilisons la commande de format '%03d' pour dire « Afficher le nombre sur une largeur de 3 caractères et, si nécessaire, complétée avec des zéros à gauche ».

```
>>> pi = 3.14159
>>> print('PI = %5.3f.' % pi)
PI = 3.142.
```

Ici, nous utilisons l'option de format en virgule flottante. Le '%5.3f' dit de produire une sortie d'une largeur totale de cinq avec trois décimales. Notez que le point décimal occupe l'un des emplacements de la largeur totale.

Une autre chose que vous pourriez ne pas avoir réalisé est que vous pouvez utiliser les clés d'un dictionnaire dans le cadre de la commande format.

```
>>> info =
{“Prenom”:“Fred”,”Nom”:“Farke
l”,”Ville”:“Denver”}
>>> print('Bienvenue
%(Prenom)s %(Nom)s de
%(Ville)!' % info)
```

```
Bienvenue Fred Farkel de
Denver !
```

Le tableau ci-dessous présente les différentes clés de substitution possibles et leurs significations.

Python 3.x

Avec Python 3.x, nous avons beaucoup plus d'options (souvenons-nous que nous pouvons les utiliser dans Python 2.7) quand il s'agit de la fonction print. Pour vous rafraîchir la mémoire,

Conversion	Meaning
'd'	Signed integer decimal
'i'	Signed integer decimal
'u'	Obsolete - identical to 'd'
'o'	Signed octal value
'x'	Signed hexadecimal - lowercase
'X'	Signed hexadecimal - uppercase
'f'	Floating point decimal
'e'	Floating point exponential - lowercase
'E'	Floating point exponential - uppercase
'g'	Floating point format - uses lowercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise
'G'	Floating point format - uses uppercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise
'c'	Single character
'r'	String (converts valid Python object using repr())
's'	String (converts valid Python object using str())
'%'	No argument is converted, results in a '%' character

voici un exemple simple de la fonction print en 3.x :

```
>>> print('{0}
{1}'.format("Salut","Python")
 )

Salut Python

>>> print("Python est {0}
sympa !".format("SUPER"))

Python est SUPER sympa !
```

Les champs de remplacement sont enfermés dans des accolades "{" "}".

Tout ce qui est en dehors de celles-ci est considéré comme littéral, et sera imprimé tel quel. Dans le premier exemple, on a numéroté les champs de remplacement 0 et 1. Cela indique à Python de prendre la première (0) valeur et la mettre dans le champ {0} et ainsi de suite. Cependant, vous n'avez pas à utiliser les numéros du tout. Utiliser cette option implique que la première valeur soit placée dans la première série de crochets et ainsi de suite.

```
>>> print("Cette version de
{} est
{}".format("Python","3.3.2"))

Cette version de Python est
3.3.2
```

Comme ils le disent sur les publicités télévisées, « MAIS ATTENDEZ... IL Y A PLUS ». Si nous avions voulu faire une mise en forme en ligne, nous avons les options suivantes :

```
:<x alignement à gauche avec
 une largeur de x
:>x alignement à droite avec
 une largeur de x
:^x alignement centré avec
 une largeur de x
```

Voici un exemple :

```
>>>
print("|{:<20}|".format("A
gauche"))
|A gauche             |
 >>>
print("|{:>20}|".format("A
droite"))
|             A droite|
 >>>

print("|{:^20}|".format("Cent
ré"))
|       Centré        |
```

Vous pouvez même spécifier un caractère de remplissage avec la justification ou la largeur :

```
>>>
print("{:*>10}".format(321.40
))
  *****321.4
```

Si vous avez besoin de formater une sortie de date ou d'heure, vous pouvez faire quelque chose comme ceci :

```
>>> d =
datetime.datetime(2013,10,9,10,
45,1)

>>>
print("{:%d/%m/%y}".format(d)
)
09/10/13

>>>
print("{:%H:%M:%S}".format(d)
)
10:45:01
```

Afficher le séparateur des milliers en utilisant une virgule (ou tout autre caractère) est simple.

```
>>> print("Voici un grand
nombre
{:.}".format(7219219281))
Voici un grand nombre
7.219.219.281
```

Eh bien, cela devrait vous donner matière à réflexion pendant assez de ce mois.

Je vous revois au début de la 5e année.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Le Podcast Ubuntu couvre toutes les dernières nouvelles et les problèmes auxquels sont confrontés les utilisateurs de Linux Ubuntu et les fans du logiciel libre en général. La séance s'adresse aussi bien au nouvel utilisateur qu'au plus ancien codeur. Nos discussions portent sur le développement d'Ubuntu, mais ne sont pas trop techniques. Nous avons la chance d'avoir quelques supers invités, qui viennent nous parler directement des derniers développements passionnants sur lesquels ils travaillent, de telle façon que nous pouvons tous comprendre ! Nous parlons aussi de la communauté Ubuntu et de son actualité.

Le podcast est présenté par des membres de la communauté Ubuntu Linux du Royaume-Uni. Il est couvert par le Code de Conduite Ubuntu et est donc adapté à tous.

L'émission est diffusée en direct un mardi soir sur deux (heure anglaise) et est disponible au téléchargement le jour suivant.

podcast.ubuntu-uk.org