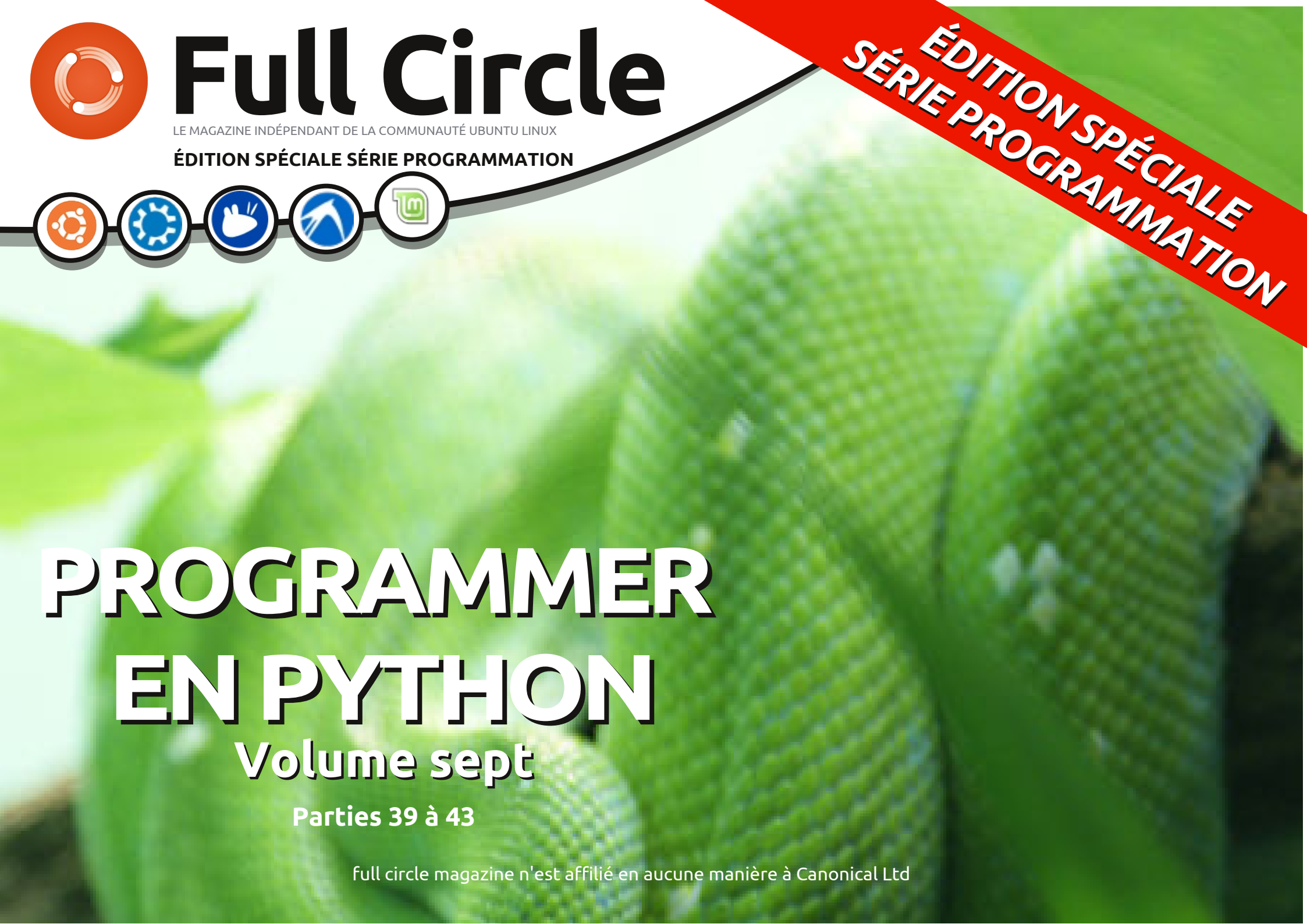
Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION



PROGRAMMER EN PYTHON

Volume sept

Parties 39 à 43

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 39 à 43**, des numéros 68 à 72, 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 !

TUTORIEL

Programmer en Python - Partie 39

Greg Walters

Il y a plusieurs longs mois, nous avons travaillé avec des appels d'API pour Weather Underground. En fait, c'était dans la partie 11, parue dans le numéro 37. Eh bien, nous allons à nouveau traiter d'API, cette fois pour un site web nommé TVRage (http://tvrage.com). Si vous ne connaissez pas ce site, il traite des émissions de télévision. Jusqu'à présent, toutes les émissions télévisées auxquelles je pouvais penser étaient dans leur système. Dans cette série d'articles, nous allons revenir sur XML, les API et ElementTree pour créer une couche d'abstraction qui nous permettra de créer une petite bibliothèque qui simplifiera notre recherche d'information sur nos émissions préférées.

Bon, j'ai mentionné une couche d'abstraction. Qu'est-ce que c'est ? En termes simples, lorsque vous créez ou utilisez une bibliothèque, vous utilisez des morceaux de code qui « enveloppent » la complexité de l'API du site Web dans une bibliothèque facile à utiliser. Avant de commencer, je dois éclaircir quelques petites choses. Tout d'abord, il s'agit d'un service gratuit. Cependant, ils demandent des dons pour l'utilisation de leur API. Si vous pensez que c'est un service utile, prière d'envisager de faire un don de 8 € ou plus. Deuxièmement, vous devez vous inscrire sur leur site et obtenir votre clé personnelle pour l'API. C'est gratuit, donc il n'y a vraiment aucune raison de ne pas le faire, surtout si vous envisagez d'utiliser les informations fournies. En outre, vous avez accès à quelques autres champs d'information comme les résumés des séries et des épisodes qui ne sont pas inclus dans la version sans inscription. Troisièmement, ils sont à pied d'œuvre sur la mise à jour de l'API. Cela signifie que quand vous lirez le présent article, leur API peut avoir changé. Nous allons utiliser les flux publics, qui sont gratuits pour tout le monde depuis décembre 2012. Le site de l'API est situé à http://services.tvrage.com/info.php?page=main et montre quelques exemples des types d'informations qui sont disponibles.

Bon, commençons à regarder l'API pour voir comment nous pouvons l'utiliser.

En utilisant leur API, nous pouvons obtenir des informations très précises sur l'émission elle-même et/ou sur chacun des épisodes. Il faut en fait trois étapes pour trouver des informations sur une émission. Les voici :

- Rechercher dans leur base de données le nom de l'émission pour obtenir son identifiant (« Show ID ») que l'on doit utiliser pour obtenir plus de données. Pensez à la valeur showid comme une clé permettant d'accéder directement à un enregistrement dans la base de données ; c'est exactement ça.
- Une fois que vous avez l'ID, obtenez les informations sur l'émission.
- Enfin, recherchez les informations sur un épisode spécifique. Elles viendront d'une liste des épisodes de l'émission jusqu'à aujourd'hui.

Nous utiliserons trois appels Web de base pour obtenir cette information. Le premier pour la recherche, le deuxième pour les informations sur l'émission et le dernier pour obtenir la liste des épisodes.

Voici les appels de base que nous allons utiliser :

- Recherche de l'identifiant ShowID à partir du nom de l'émission - `http://services.tvrage.com/feeds/search.php?show={UneEmission}`
- Récupération des données de l'émission à partir du ShowID (sid) - `http://services.tvrage.com/feeds/showinfo.php?sid={UnIdentifiant}`
- Récupération de la liste des épisodes pour ShowID (sid) - `http://services.tvrage.com/feeds/episode_list.php?sid={UnIdentifiant}`

On reçoit en retour un flux de données au format XML. Prenons un moment pour revoir à quoi ressemble XML. La première ligne doit toujours être similaire à celle illustrée ci-dessous pour être considérée comme un bon flux de données XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<BALISE RACINE>
  <BALISE PARENT>
    <BALISE ENFANT 1>DONNÉES</FIN DE BALISE ENFANT 1>
    <BALISE ENFANT 2>DONNÉES</FIN DE BALISE ENFANT 2>
    <BALISE ENFANT 3>DONNÉES</FIN DE BALISE ENFANT 3>
  </FIN DE BALISE PARENT>
</FIN DE BALISE RACINE>
```

Chaque donnée est entourée par une balise de définition et une balise de fin. Parfois, vous aurez une balise enfant qui est aussi elle-même une balise parent comme ceci :

```
<BALISE ENFANT PARENT>

<BALISE ENFANT 1>DONNÉES</FIN DE BALISE ENFANT 1>

</FIN DE BALISE ENFANT PARENT>
```

Vous pouvez également trouver une balise à laquelle est associé un attribut :

```
<BALISE INFORMATION = VALEUR>

<BALISE ENFANT>DONNÉES</FIN BALISE ENFANT>

</FIN BALISE>
```

Parfois, vous pourrez voir une balise sans données associées. Cela ressemblera à ceci :

```
<prodnum/>
```

Parfois, s'il n'y a pas d'information pour une certaine balise, cette balise ne sera tout simplement pas présente. Votre programme devra faire face à ces possibilités.

Donc, pour recevoir et traiter les données XML, nous commençons par la balise racine et analysons chaque balise, à la recherche des données qui nous intéressent. Dans certains cas, nous voulons tout récupérer, dans d'autres nous nous préoccupons seulement de certains morceaux de l'information.

Maintenant, penchons-nous sur le premier appel et regardons ce qui est retourné. Supposons que l'émission que nous cherchons est Buffy contre les vampires. Notre appel de recherche devrait ressembler à ceci :

```
http://services.tvrage.com/feeds/search.php?show=buffy
```

Le fichier XML retourné devrait ressembler à ceci : http://pastebin.com/Eh6ZtJ9N.

Notez que j'ai indenté moi-même pour vous faciliter la lecture. Maintenant, nous allons décomposer le fichier XML pour voir ce qu'il contient.

<Results> - Il s'agit de la racine des données XML. La dernière ligne du flux retourné doit être la balise de fermeture </Results>. Fondamentalement, cela marque le début et la fin du flux XML. Il pourrait n'y avoir aucun résultat ou cinquante résultats.

<show> - C'est le nœud parent qui dit : « Ce qui suit (jusqu'à la balise fermante) est l'information sur une émission de télévision ». Encore une fois, il se termine par la balise de fin </show>. Tout ce qui est entre ces deux balises concerne une émission.
<showid>2930</showid> - Cette balise showid contient le sid que nous devons utiliser pour obtenir les informations de l'émission, dans ce cas 2930.
<name>Buffy the Vampire Slayer</name> - C'est le nom de l'émission.
<link>...</link> - C'est le lien vers l'émission elle-même (ou vers l'épisode dans le cas d'un épisode) sur le site TVRage.
<country>...</country> - Le pays d'origine de l'émission.
...
</show>
</Results>

Dans le cas de notre programme, nous ne sommes vraiment intéressés que par les deux balises <showid> et <name>. Nous pourrions également envisager de prêter attention au champ <started>. Ceci, parce que nous recevons rarement un seul ensemble de données, surtout si l'on ne donne pas le nom absolument complet de l'émission. Par exemple, si nous nous étions intéressés à l'émission « The Big Bang Theory » et l'avions recherchée en utilisant uniquement la chaîne « Big Bang », nous aurions obtenu une vingtaine d'ensembles de données en retour car tout ce qui s'approche, même de loin, de « big » ou de « bang » serait renvoyé. De même, si nous nous étions intéressés à l'émission "NCIS" et avions cherché cela, nous trouverions de nombreuses réponses. Certaines ne correspondant pas à ce que nous attendons. Non seulement nous obtenons « NCIS », « NCIS : Los Angeles », « The Real NCIS », mais aussi «_Les rues de San Francisco » et « L'enquête Da Vinci », et beaucoup plus, puisque les lettres « N » « C » « I » et «_S » sont contenus dans tous ceux-ci, à peu près dans cet ordre.

Une fois que nous connaissons l'identifiant d'émission, nous pouvons demander l'information sur l'émission

pour cet ID-là. Les données sont similaires à celles que nous venons de recevoir dans la réponse à la recherche, mais plus détaillées. En utilisant encore une fois Buffy comme exemple, voici (page suivante, à droite) une version abrégée du fichier XML.

Vous pouvez voir que la plupart des données étaient déjà dans le flux de réponse à la recherche originale. Cependant, des choses comme la chaîne, le pays de la chaîne, l'exécution, le jour et l'heure de diffusion, sont spécifiques à cette série de réponses.

Ensuite, nous allons demander la liste des épisodes. Si l'émission n'a qu'une saison et a/avait seulement six épisodes, ce flux sera court. Cependant, prenons le cas d'une de mes émissions préférées, Doctor Who. Doctor Who est une émission britannique qui, dans sa forme originale, a commencé en 1963 et a duré 26 saisons jusqu'en 1989. Sa première saison compte 42 épisodes, tandis que les autres saisons/séries ont environ 24 épisodes. Vous pouvez voir que vous pourriez avoir un ÉNORME flux à analyser.

Ce que nous obtenons après la demande de la liste des épisodes est indiqué sur la page suivante (en utilisant encore Buffy comme exemple) ; je vais juste utiliser une partie du flux pour que vous ayez une bonne idée de ce qu'on reçoit.

Pour résumer, donc, l'information dont nous avons vraiment besoin après la recherche de l'identifiant de l'émission à partir de son nom serait :

```
<showid>
<name>
<started>
```

Dans le flux d'information sur l'émission, nous voudrions (normalement) :

```
<seasons>
<started>
<start date>
<origin_country>
<status>
<genres>
<runtime>
<network>
<airtime>
<airday>
<timezone>
```

et dans le flux de la liste des épisodes :

```
<Season>
<episode number>
<season number>
<production number>
<airdate>
<link>
<title>
```

```
<Showinfo>
   <showid>2930</showid>
   <showname>Buffy the Vampire Slayer</showname>
   <showlink>http://tvrage.com/Buffy_The_Vampire_Slayer</showlink>
   <seasons>7</seasons>
   <started>1997</started>
   <startdate>Mar/10/1997</startdate>
   <ended>May/20/2003</ended>
   <origin_country>US</origin_country>
   <status>Canceled/Ended</status>
   <classification>Scripted</classification>
   <genres>
      <genre>Action</genre>
      <genre>Adventure</genre>
      <genre>Comedy</genre>
      <genre>Drama</genre>
      <genre>Mystery</genre>
      <genre>Sci-Fi</genre>
   </genres>
   <runtime>60</runtime>
   <network  country="US">UPN</network>
   <airtime>20:00</airtime>
   <airday>Tuesday</airday>
   <timezone>GMT-5 -DST</timezone>
   <akas>
      <aka country="SE">Buffy &amp;amp; vampyrerna</aka>
      <aka country="DE">Buffy - Im Bann der Dämonen</aka>
      <aka country="NO">Buffy - Vampyrenes skrekk</aka>
      <aka country="HU">Buffy a vámpírok réme</aka>
      <aka country="FR">Buffy Contre les Vampires</aka>
      <aka country="IT">Buffy l'Ammazza Vampiri</aka>
      <aka country="PL">Buffy postrach wampirów</aka>
      <aka country="BR">Buffy, a Caça-Vampiros</aka>
      <aka country="PT">Buffy, a Caçadora de Vampiros</aka>
      <aka country="ES">Buffy, Cazavampiros</aka>
      <aka country="HR">Buffy, ubojica vampira</aka>
      <aka country="FI">Buffy, vampyyrintappaja</aka>
      <aka country="EE">Vampiiritapja Buffy</aka>
      <aka country="IS">Vampírubaninn Buffy</aka>
   </akas>
</Showinfo>
```

Un petit « avertissement » ici. Le numéro de saison et les numéros des épisodes ne sont pas forcément ce à quoi vous pensez. Dans le cas des données de TVRage, le numéro de saison est le numéro de l'épisode dans la saison. Le numéro d'épisode est le numéro de cet épisode dans la durée de vie totale de la série. Le numéro de production est un numéro qui a été utilisé en interne pour la série et qui, pour beaucoup de gens, n'a pas vraiment de signification.

Maintenant que nous avons rafraîchi notre mémoire sur la structure des fichiers XML et examiné les appels à l'API de TVRage, nous sommes prêts à commencer à coder, mais cela devra attendre jusqu'à la prochaine fois.

Jusque-là, passez de bonnes vacances.

```
<Show>
   <name>Buffy the Vampire Slayer</name>
   <totalseasons>7</totalseasons>
   <Episodelist>
      <Season no="1">
         <episode>
            <epnum>1</epnum>
            <seasonnum>01</seasonnum>
            <prodnum>4V01</prodnum>
            <airdate>1997-03-10</airdate>
            <link>http://www.tvrage.com/Buffy_The_Vampire_Slayer/episodes/28077</link>
            <title>Welcome to the Hellmouth (1)</title>
         </episode>
         <episode>
            <epnum>2</epnum>
            <seasonnum>02</seasonnum>
            <prodnum>4V02</prodnum>
            <airdate>1997-03-10</airdate>
            <link>http://www.tvrage.com/Buffy_The_Vampire_Slayer/episodes/28078</link>
            <title>The Harvest (2)</title>
         </episode>
         <episode>
            <epnum>3</epnum>
            <seasonnum>03</seasonnum>
            <prodnum>4V03</prodnum>
            <airdate>1997-03-17</airdate>
            <link>http://www.tvrage.com/Buffy_The_Vampire_Slayer/episodes/28079</link>
            <title>Witch</title>
         </episode>
         ...
      </Season>
   </Episodelist>
</Show>
```

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

TUTORIEL

Greg Walters

Programmer en Python - Partie 40

La dernière fois, nous avons eu une longue discussion à propos de l'API web TVRAGE. Cette fois-ci, nous allons commencer à écrire du code et à nous en servir.

Le but de cette partie est de commencer le processus de création de code qui sera un module réutilisable pouvant être importé dans un autre programme python et qui donnera accès à l'API facilement.

Bien que l'API TVRAGE nous fournisse un certain nombre de possibilités, a fortiori pour la version enregistrée, nous allons nous concentrer sur seulement trois appels :
1. Rechercher une émission par son nom et obtenir le ShowID.
2. Obtenir de l'information sur l'émission à partir du ShowID.
3. Obtenir des informations spécifiques à un épisode à partir du ShowID.

La dernière fois, je vous ai montré les appels de l'API « non enregistrée » qui sont accessibles par tout le monde. Cette fois, nous allons utiliser les appels enregistrés – basés sur une clé d'enregistrement que j'ai. Je vais partager avec vous cette clé (TVRAGE sait que je vais le faire). Cependant, je vous demande, s'il vous plaît, si vous envisagez d'utiliser l'API, de vous inscrire et d'obtenir votre propre clé, pour ne pas abuser du site. Je vous saurais gré de réfléchir également à leur faire un don pour soutenir leurs efforts cons-tants.

Nous allons créer trois programmes principaux pour faire les appels et retourner l'information, trois routines qui seront utilisées pour afficher les informations retournées (en supposant que nous sommes en mode « indépendant »), et un sous-programme principal pour faire le travail – en supposant, là encore, que nous sommes en mode « indépendant ».

Voici la liste des routines que nous allons créer (enfin pas toutes pour cette fois-ci. Je veux laisser la place à d'autres choses dans ce numéro).

```
def TrouverIdParNom(self,
nomEmission, debug = 0)

def
RecupererInformationEmission(
self, showid, debug = 0)

def
RecupererListeEpisodes(self,
showid, debug = 0)
def
AfficheResultatsEmission(self,
ListeEmissionsDict)
def
AfficheInformationEmission(se
lf, dict)

def
AfficheListeEpisodes(self,
NomsEmission, NumeroSaison,
ListeEpisodes)

def main()
```

La routine TrouverIdParNom prend une chaîne (nomEmission), effectue l'appel API, analyse la réponse XML et retourne une liste des émissions qui correspondent aux informations contenues dans un dictionnaire ; ainsi, ce sera une liste de dictionnaires. RecupererInformationEmission récupère le showid de la routine précédente et retourne un dictionnaire d'informations sur l'émission. RecupererListeEpisodes utilise également le showid de la routine ci-dessus et retourne une liste de dictionnaires contenant des informations pour chaque épisode.

Nous utiliserons une série de chaînes pour contenir la clé et l'URL de base, puis leur ajouter ce dont nous avons besoin. Par exemple, considérons le code suivant (nous le compléterons plus tard).

```
self.CleApi =
"Itnl8IyY1hsR9n0IP6zI"
self.ChaineRechercheSerie =
"http://services.tvrage.com/m
yfeeds/search.php?key="
```

L'appel que nous devons envoyer (pour récupérer une liste d'informations sur la série avec l'id de la série) serait :

http://services.tvrage.com/myfeeds/search.php?key=Itnl8IyY1hsR9n0IP6zI&show={NomEmission}

Nous combinons la chaîne comme ceci :

```
chaine =
self.ChaineRechercheSerie +
self.CleAPI + "&show=" +
nomEmission
```

Pour les besoins des tests, je vais utiliser une série intitulée « Continuum » qui, si vous ne l'avez jamais vue, est une série géniale de science-fiction sur la chaîne canadienne Showcase. J'utilise cette série pour plusieurs raisons. Tout d'abord, il n'y a (lorsque j'écris ces lignes) que deux séries qui correspondent à la recherche « Continuum », ce qui rend votre débogage facile, et, d'autre part, il n'y a actuellement qu'une seule saison de 10 épisodes à gérer.

Vous devriez avoir une idée de ce que vous rechercherez dans vos routines d'analyse ; j'ai donc placé ci-dessous les URL complètes pour que vous les testiez avant de vous lancer dans votre code.

Rechercher en utilisant un nom d'émission :
http://services.tvrage.com/myfeeds/search.php?key=Itnl8IyY1hsR9n0IP6zI&show=continuum

Récupérer des informations sur la série avec le ShowID (sid) :
http://services.tvrage.com/myfeeds/showinfo.php?key=Itnl8IyY1hsR9n0IP6zI&sid=30789

Récupérer la liste des épisodes et leurs informations avec le ShowID (sid) :
http://services.tvrage.com/myfeeds/episode_list.php?key=Itnl8IyY1hsR9n0IP6zI&sid=30789

Maintenant que nous avons vu tout cela, nous allons commencer à écrire le code.

Vous allez créer un fichier nommé « tvrage.py ». Nous allons nous en servir pendant un ou deux articles.

Nous allons commencer avec nos importations indiquées en haut à droite.

Vous pouvez voir que nous allons utiliser ElementTree pour faire l'analyse XML et urllib pour la communication internet. La bibliothèque sys est utilisée pour sys.exit.

Nous allons mettre en place la boucle principale maintenant afin de pouvoir tester les choses au fur et à mesure (ci-dessous). Rappelez-vous que ceci doit être tout à la fin de notre fichier source.

```
#=========================================================
#                      IMPORTS
#=========================================================
from xml.etree import ElementTree as ET
import urllib
import sys
```

Comme je l'ai dit plus tôt, les quatre premières lignes sont nos chaînes partielles pour construire l'URL de la fonction que nous voulons utiliser. (ChaineListeEpisodes doit être sur une seule ligne.) Les quatre dernières lignes sont l'initialisation des listes que nous utiliserons plus tard.

Tout d'abord (page précédente au milieu à droite), nous réglons la chaîne qui sera utilisée comme URL. Puis, nous

```
    def TrouverIdParNom(self,nomEmission,debug = 0):
        chaine = self.ChaineRechercheSerie + self.CleAPI + "&show=" + nomEmission
        urllib.socket.setdefaulttimeout(8)
        usock = urllib.urlopen(chaine)
        resultat = ET.parse(usock).getroot()
        usock.close()
        compteurTrouves = 0
        self.listeEmissions = []
```

```
#=========================================================
#     Main loop
#=========================================================
if __name__ == "__main__":
    main()
```

Maintenant nous commençons notre classe. Le nom de la classe est "TvRage". Nous allons aussi faire notre routine __init__.

```
class TvRage:
    def __init__(self):
        self.CleAPI = "Itnl8IyY1hsR9n0IP6zI"
        self.ChaineRechercheSerie = "http://services.tvrage.com/myfeeds/search.php?key="
        self.ChaineInformationEmission =
"http://services.tvrage.com/myfeeds/showinfo.php?key="
        self.ChaineListeEpisodes =
"http://services.tvrage.com/myfeeds/episode_list.php?key="
        self.ListeEmissions = []
        self.InfosEmissions = []
        self.ListeEpisodes = []
```

réglons le socket avec un délai d'attente de 8 secondes par défaut. Ensuite, nous appelons urllib.urlopen avec l'URL générée et (espérons-le) recevons notre fichier xml dans l'objet usock. Nous utilisons ElementTree pour analyser les informations xml. (Si vous êtes perdus, relisez s'il vous plaît mes articles sur XML (les parties 10, 11 et 12 figurant dans les FCM n° 36, 37 et 38)). Enfin, on ferme le socket et on initialise le compteur pour le nombre de résultats trouvés, puis on réinitialise la liste listeEmissions à une liste vide.

Maintenant, nous allons passer en revue les informations XML en utilisant la balise « show » comme parent de ce que nous voulons. Rappelez-vous que les informations retournées ressemblent à ce qui est en haut à droite.

Nous allons parcourir chaque groupe d'informations en cherchant « show » comme parent et analyser l'information. En pratique, nous n'avons besoin que du nom de l'émission (<name>) et du showid (<showid>) montré en bas à gauche, mais nous allons gérer tous les résultats.

```
for noeud in
resultat.findall('show'):
    infosEmissions = []
    chaineGenre = None
    dict = {}
    for n in noeud:
        if n.tag == 'showid':
            showid = n.text
            dict['ID'] = showid
```

Je vais expliquer le premier et vous comprendrez le reste. Lorsque nous parcourons les informations, nous recherchons les balises (en bas à droite) qui correspondent à ce que nous voulons. Si nous en trouvons une, nous l'affectons à une variable temporaire, puis mettons cela dans le dictionnaire comme valeur avec une clé qui correspond à ce que nous insérons. Dans le cas qui précède, nous recherchons la balise « showid » dans les données XML. Lorsque nous la trouvons, nous l'assignons en tant que valeur de la clé « ID ».

La partie suivante (page suivante, en haut à droite) porte sur le(s) genre(s) de la série. Comme vous pouvez le voir dans l'extrait XML ci-dessus, cette série se trouve dans quatre genres différents. Action, crime, drame et Sci-Fi. Nous devons les traiter tous.

Enfin, on incrémente la variable compteurTrouves et on ajoute ce dictionnaire dans la liste « listeEmissions ». Ensuite, nous recommençons le processus jusqu'à ce qu'il n'y ait plus aucune donnée XML. Une fois que tout est terminé, on retourne la liste des dictionnaires (en bas à droite).

```
<Results>
    <show>
        <showid>30789</showid>
        <name>Continuum</name>
        <link>http://www.tvrage.com/Continuum</link>
        <country>CA</country>
        <started>2012</started>
        <ended>0</ended>
        <seasons>2</seasons>
        <status>Returning Series</status>
        <classification>Scripted</classification>
        <genres>
            <genre>Action</genre>
            <genre>Crime</genre>
            <genre>Drama</genre>
            <genre>Sci-Fi</genre>
        </genres>
    </show>
    ...
</Results>
```

```
        elif n.tag == 'name':
            nomEmission = n.text
            dict['Nom'] = nomEmission
        elif n.tag == 'link':
            showlink = n.text
            dict['Lien'] = showlink
        elif n.tag == 'country':
            showcountry = n.text
            dict['Pays'] = showcountry
        elif n.tag == 'started':
            showstarted = n.text
            dict['Debut'] = showstarted
        elif n.tag == 'ended':
            showended = n.text
            dict['Fin'] = showended
        elif n.tag == 'seasons':
            showseasons = n.text
            dict['Saisons'] = showseasons
        elif n.tag == 'status':
            showstatus = n.text
            dict['Etat'] = showstatus
        elif n.tag == 'classification':
            showclassification = n.text
            dict['Classification'] = showclassification
```

La plupart du code est assez explicite. Nous allons nous concentrer sur la boucle « for » que nous utilisons pour afficher les informations. Nous bouclons sur chaque élément de la liste de dictionnaires et affichons une variable compteur, le nom de l'émission (c['Nom']) et l'id. Le résultat ressemble à ceci :

```
Entrer le nom de la série → continuum
2 resultat(s)
-----------------------
1 - Continuum - 30789
2 - Continuum (Web series) - 32083
Choisir un nombre ou 0 pour quitter →
```

Souvenez-vous que la liste des articles commence à zéro, donc lorsque l'utilisateur entre 1, en fait il demande le dictionnaire numéro 0. Nous faisons comme ça parce que les gens « normaux » pensent que le décompte doit commencer par 1, pas par 0. Et nous pouvons ainsi utiliser 0 pour quitter la routine et ne pas leur faire utiliser Q ou q ou -1.

Maintenant, la routine « main » va tout rassembler pour nous.

Pour aujourd'hui, nous allons juste commencer la routine (au milieu à droite) et nous la continuerons la prochaine fois.

La prochaine fois, nous ajouterons les autres routines. Pour l'instant, le code peut être trouvé sur http://pastebin.com/8F3Bd1Xd

À 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.

```
elif n.tag == 'genres':
    for sousElement in n:
        if sousElement.tag == 'genre':
            if sousElement.text != None:
                if chaineGenre == None:
                    chaineGenre = sousElement.text
                else:
                    chaineGenre += " | " + sousElement.text
    dict['Genres'] = chaineGenre
```

```
def main():
    tr = TvRage()
    #--------------------
    # Chercher une serie par son nom
    #--------------------
    nom = raw_input("Entrer le nom de la serie -> ")
    if nom != None:
        liste = tr.TrouverIdParNom(nom)
        choix = tr.AfficheResultatsEmission(liste)
        print "choix %d" % int(choix)

        if int(choix) == 0:
            sys.exit()
        else:
            option = int(choix)-1
            id = liste[option]['ID']
            print "Le ShowID choisi est %s" % id
```

```
            compteurTrouves += 1
            self.listeEmissions.append(dict)
        return self.listeEmissions
#=======================================================
```

La prochaine chose que nous allons faire est de créer la routine pour afficher l'ensemble de nos résultats.

```
    def AfficheResultatsEmission(self, ListeEmissionsDict):
        tailleListe = len(ListeEmissionsDict)
        print "%d resultat(s)" % tailleListe
        print "-----------------------"
        compteur = 1
        for c in ListeEmissionsDict:
            print "%d - %s - %s" % (compteur,c['Nom'],c['ID']) #,c['Fin'],c['Lien']
            compteur += 1
        sel = raw_input("Choisir un nombre ou 0 pour quitter -> ")
        return sel
```

TUTORIEL

Programmer en Python - Partie 41

Greg Walters

Le mois dernier, nous avons commencé notre version en ligne de commande d'une bibliothèque pour discuter avec l'API Web TVRage. Ce mois-ci, nous allons continuer à ajouter du code à cette bibliothèque. Si vous n'avez pas le code du mois dernier, veuillez le récupérer sur pastebin (http://pastebin.com/8F3Bd1Xd) puisque que nous allons compléter ce code.

Dans l'état où nous avons laissé le code, vous devez exécuter le programme et entrer dans la fenêtre du terminal le nom d'une émission de télévision pour laquelle vous souhaitez obtenir des informations. Rappelez-vous, nous avons utilisé la série Continuum. Une fois que vous avez appuyé sur <Entrée>, le programme appelait l'API et faisait une recherche avec le nom de l'émission, puis renvoyait une liste de noms d'émissions correspondant à votre saisie. Vous pouviez ensuite sélectionner dans la liste en entrant un numéro et il affichait « le ShowID choisi est 30789 ». Maintenant, nous allons écrire le code qui va utiliser ce ShowID pour obtenir les informations sur la série. Une autre chose à garder à l'esprit : les routines d'affichage sont là simplement pour prouver que les routines fonctionnent. Le but ultime est de créer une bibliothèque réutilisable qui peut être utilisée dans un programme graphique. N'hésitez pas à modifier les routines d'affichage si vous voulez faire plus avec les capacités autonomes de la bibliothèque.

```
    def TrouveListeEpisodes(self,idemission,debug=0):
        idemissionchaine = str(idemission)
        chaine = self.ChaineListeEpisodes + self.CleAPI + "&sid=" +
idemissionchaine
        urllib.socket.setdefaulttimeout(8)
        usock = urllib.urlopen(chaine)
        arbre = ET.parse(usock).getroot()
        usock.close()
        dict = {}
```

La dernière routine que nous avons créée dans la classe était « AfficheResultatsEmission ». Nous allons placer notre prochaine routine juste après, et avant la routine « main ». L'information qui sera retournée (il y en a d'autres, mais nous allons utiliser uniquement la liste ci-dessous) sera dans un dictionnaire et contiendra (si disponible) :

- identifiant de l'émission ;
- nom de l'émission ;
- lien de l'émission ;
- pays d'origine ;
- nombre de saisons ;
- image de la série ;
- année de démarrage ;
- date de démarrage ;
- date de fin ;
- état (annulé, rediffusion, actuel, etc.) ;
- classification (fiction, réalité, etc.) ;
- résumé de la série ;
- genre(s) ;
- durée en minutes ;
- nom de la chaîne qui a diffusé l'émission pour la première fois ;
- pays de la chaîne (c'est à peu près la même chose que pays d'origine) ;
- heure de diffusion ;
- jour de diffusion (dans la semaine) ;
- fuseau horaire.

Ci-dessus, le début du code.

Vous devez reconnaître la plupart du code de la dernière fois. Il n'a vraiment pas beaucoup changé. Voici plus de code (voir ci-dessous).

```
        for enfant in arbre:
            if enfant.tag == 'showid':
                dict['ID'] = enfant.text
            elif enfant.tag == 'showname':
                dict['Nom'] = enfant.text
            elif enfant.tag == 'showlink':
                dict['Lien'] = enfant.text
            elif enfant.tag == 'origin_country':
                dict['Pays'] = enfant.text
            elif enfant.tag == 'seasons':
                dict['Saisons'] = enfant.text
            elif enfant.tag == 'image':
                dict['Image'] = enfant.text
            elif enfant.tag == 'started':
                dict['Debut'] = enfant.text
            elif enfant.tag == 'startdate':
                dict['DateDebut'] = enfant.text
```

```
        elif enfant.tag == 'ended':
            dict['Fin'] = enfant.text
        elif enfant.tag == 'status':
            dict['Etat'] = enfant.text
        elif enfant.tag == 'classification':
            dict['Classification'] = enfant.text
        elif enfant.tag == 'summary':
            dict['Resume'] = enfant.text
```

Comme vous pouvez le voir (ci-dessus), il n'y a rien de vraiment nouveau dans ce morceau de code non plus, si vous avez suivi la série. Nous utilisons une boucle for pour vérifier chaque balise dans le fichier XML par rapport à une valeur spécifique. Si nous la trouvons, nous l'assignons à une entrée du dictionnaire.

À présent, les choses se compliquent un peu. Nous allons chercher la balise « genres ». Elle a des balises enfants en dessous d'elle avec le nom de « genre ». Pour un spectacle donné, il peut y avoir plusieurs genres. Nous devrons ajouter les genres à une chaîne au fur et à mesure qu'ils arrivent et les séparer par une barre verticale et deux espaces comme ceci " | " (voir en haut à droite).

Maintenant, nous sommes à peu près revenus au code « normal » (affiché au milieu à droite) que vous avez déjà vu. La seule chose un peu différente, c'est le tag « chaîne » qui a un attribut « pays ». Nous récupérons les données d'attribut par la recherche de « child.attrib['attributetag'] » au lieu de « child.text ».

C'est la fin de cette routine. Maintenant (ci-dessous), nous avons besoin d'une méthode pour afficher les informations que nous avons obtenues par ce si dur travail. Nous allons créer une routine appelée « AfficheInfoEmission ».

Maintenant, nous devons mettre à jour la routine « main » (page suivante, en haut à droite) pour prendre en compte nos deux nouvelles routines. Je donne la routine entière ci-dessous, mais le nouveau code est affiché en noir.

```
elif enfant.tag == 'genres':
    chainegenre = None
    for souselement in enfant:
        if souselement.tag == 'genre':
            if souselement.text != None:
                if chainegenre == None:
                    chainegenre = souselement.text
                else:
                    chainegenre += " | " + souselement.text
    dict['Genres'] = chainegenre
```

```
      elif enfant.tag == 'runtime':
          dict['Diffusion'] = enfant.text
      elif enfant.tag == 'network': # a un attribut
          #print enfant.attrib['country'],enfant.text
          dict['PaysDiffusion'] = enfant.attrib['country']
          dict['Chaine'] = enfant.text
      elif enfant.tag == 'airtime':
          dict['HeureDiffusion'] = enfant.text
      elif enfant.tag == 'airday':
          dict['JourDiffusion'] = enfant.text
      elif enfant.tag == 'timezone':
          dict['FuseauHoraire'] = enfant.text
  return dict
```

En bas à gauche de la page suivante, on voit à quoi devrait ressembler la sortie de « AfficheInfoEmission », en supposant que vous avez choisi « Continuum » comme émission.

Veuillez noter que je n'inclus pas l'affichage des informations de fuseau

```
    def AfficheInfoEmission(self,dict):
        print "Emission : %s" % dict['Nom']
        print "ID : %s  Debut : %s  Fin : %s  Date debut : %s  Saisons : %s" %
(dict['ID'],dict['Debut'],dict['Fin'],dict['DateDebut'],dict['Saisons'])
        print "Lien : %s" % dict['Lien']
        print "Image : %s" % dict['Image']
        print "Pays : %s  Etat : %s   Classification : %s" %
(dict['Pays'],dict['Etat'],dict['Classification'])
        print "Diffusion : %s  Chaine : %s   Jour diffusion : %s   Heure diffusion : %s"
% (dict['Diffusion'],dict['Chaine'],dict['JourDiffusion'],dict['HeureDiffusion'])
        print "Genres : %s" % dict['Genres']
        print "Resume : \n%s" % dict['Resume']
```

horaire ici, mais n'hésitez pas à l'ajouter si vous le souhaitez.

Ensuite, nous devons travailler sur la routine qui liste les épisodes pour la série. La routine « qui travaille » sera appelée « TrouveListeEpisodes » et fournira les informations suivantes :

- Saison ;
- numéro de l'épisode ;
- numéro de l'épisode au sein de la saison ;
- numéro de production ;
- date de diffusion ;
- lien ;
- titre ;
- résumé ;
- évaluation ;
- capture d'image de l'épisode (si disponible).

```
ShowID selected was 30789
Show: Continuum
ID: 30789  Started: 2012  Ended: None  Start Date:
May/27/2012  Seasons: 2
Link: http://www.tvrage.com/Continuum
Image: http://images.tvrage.com/shows/31/30789.jpg
Country: CA  Status: Returning Series  Classification:
Scripted
Runtime: 60  Network: Showcase  Airday: Sunday
Airtime: 21:00
Genres: Action | Crime | Drama | Sci-Fi
Contenu :
Continuum est une série dramatique policière d'une heure,
centrée sur Kiera Cameron, une femme flic ordinaire qui vient
de l'an 2077, et qui se retrouve prise au piège dans
l'actuelle Vancouver. Elle est seule, une étrangère dans un
pays étranger, et traque huit des criminels les plus
impitoyables venus du futur, appelés Liber8, qui rôdent dans
la ville.

Heureusement pour Kiera, grâce à l'utilisation de son CMR
(rappel de mémoire cellulaire), une technologie à puce liquide
futuriste implantée dans son cerveau, elle communique avec
Alec Sadler, un génie technologique de dix-sept ans. Lorsque
Kiera l'appelle et Alec lui répond, un partenariat unique en
son genre commence.

Le souhait principal de Kiera est de rentrer « à la maison ».
Mais, jusqu'à ce qu'elle comprenne comment le faire, elle doit
survivre à notre époque et utiliser toutes les ressources à sa
disposition pour suivre et capturer les terroristes avant
qu'ils n'altèrent l'histoire suffisamment pour changer
l'avenir. Après tout, à quoi bon y retourner si l'avenir
n'est pas celui qui vous avez quitté ?
```

```
def main():
    tr = TvRage()
    #--------------------
    # Chercher une serie par son nom
    #--------------------
    nom = raw_input("Entrer le nom de la serie -> ")
    if nom != None:
        liste = tr.TrouverIdParNom(nom)
        choix = tr.AfficheResultatsEmission(liste)
        if int(choix) == 0:
            sys.exit()
        else:
            option = int(choix)-1
            id = liste[option]['ID']
            print "Le ShowID choisi est %s" % id
    #------------------------------
    # Recupere les informations
    #------------------------------
    infoemission = tr.TrouveInfoEmission(id)
    #------------------------------
    # Affiche les informations
    #------------------------------
    tr.AfficheInfoEmission(infoemission)
```

Avant que nous commencions avec le code, il serait utile de revenir sur ce que l'API retourne lors de la demande de la liste des épisodes. Cela ressemble à ce qui est en haut à droite de la page suivante.

Les informations pour chaque épisode sont dans la balise « épisode » – qui est un enfant de « saison » – qui est un enfant de « ListeEpisodes » – qui est un enfant de « Emission ». Nous devons faire attention à la façon dont nous analysons ceci. Comme avec la plupart de nos routines « utilitaires » de cette fois-ci, les quelques premières

```
    def TrouveListeEpisodes(self,idemission,debug=0):
        idemissionchaine = str(idemission)
        chaine = self.ChaineListeEpisodes + self.CleAPI +
"&sid=" + idemissionchaine
        urllib.socket.setdefaulttimeout(8)
        usock = urllib.urlopen(chaine)
        arbre = ET.parse(usock).getroot()
        NomEmission = ""
        SaisonsTotal = ""
        usock.close()
        for enfant in arbre:
```

```
            if enfant.tag == 'name':
                NomEmission = enfant.text
            elif enfant.tag == 'totalseasons':
                SaisonsTotal = enfant.text
            elif enfant.tag == 'Episodelist':
                for c in enfant:
                    if c.tag == 'Season':
                        dict = {}
                        numsaison = c.attrib['no']
                        for el in c:
```

lignes (page précédente en bas à droite) sont assez faciles à comprendre à présent.

Maintenant, nous devons chercher les balises « Nom » et « SaisonsTotal » en dessous de la balise racine « Emission ». Une fois que nous les avons traitées, nous chercherons les balises « ListeEpisodes » et « Saison ». Remarquez ci-dessus que la balise « Saison » a un attribut. Vous remarquerez peut-être (dans le code ci-dessus) que nous n'incluons pas les données « NomEmission » ni «SaisonsTotal » dans le dictionnaire. Nous les assignons à une variable qui sera renvoyée au code appelant à la fin de la routine.

```
<Show>
<name>Continuum</name>
<totalseasons>2</totalseasons>
<Episodelist>
<Season no="1">
<episode>
<epnum>1</epnum>
<seasonnum>01</seasonnum>
<prodnum/>
<airdate>2012-05-27</airdate>
<link>
http://www.tvrage.com/Continuum/episodes/1065162187
</link>
<title>A Stitch in Time</title>
<summary>
L'inspecteur Kiera Cameron perd tout ce qu'elle a et se retrouve
avec une nouvelle mission quand elle et huit terroristes
dangereux sont transportés de leur époque, 2077, à 2012, pendant
la tentative des terroristes d'échapper à leur exécution. Elle
prend une nouvelle identité et rejoint le VPD [Ndt : Vancouver
Police Department] afin d'arrêter le règne de violence des
terroristes. Sur le chemin, elle se lie d'amitié avec Alec
Sadler, le jeune de 17 ans qui, un jour, réussira à créer la
technologie sur laquelle son monde à elle est construit.
</summary>
<rating>8.8</rating>
<screencap>
http://images.tvrage.com/screencaps/154/30789/1065162187.png
</screencap>
</episode>
```

```
    if el.tag == 'episode':
        dict={}
        dict['Saison'] = numsaison

        for ep in el:
            if ep.tag == 'epnum':
                dict['NumeroEpisode'] = ep.text
            elif ep.tag == 'seasonnum':
                dict['NumeroEpisodeSaison'] = ep.text
            elif ep.tag == 'prodnum':
                dict['ProductionNumber'] = ep.text
            elif ep.tag == 'airdate':
                dict['DateDiffusion'] = ep.text
            elif ep.tag == 'link':
                dict['Lien'] = ep.text
            elif ep.tag == 'title':
                dict['Titre'] = ep.text
            elif ep.tag == 'summary':
                dict['Resume'] = ep.text
            elif ep.tag == 'rating':
                dict['Notation'] = ep.text
            elif ep.tag == 'screencap':
                dict['CaptureEcran'] = ep.text
```

Maintenant que nous avons cette partie des données, nous traitons les informations spécifiques à l'épisode (voir en bas à gauche).

Tout ce qui reste à faire (en bas à droite) est d'ajouter les informations spécifiques de l'épisode (que nous avons mises dans le dictionnaire) à notre liste, et de continuer. Une fois que nous avons fini avec tous les épisodes, nous revenons à la routine d'appel et, comme je l'ai dit plus tôt,

```
self.ElementEpisode.append(dict)
        return NomEmission,SaisonsTotal,self.ElementEpisode
```

retournons trois données, « NomEmission », « SaisonsTotal » et la liste des dictionnaires.

Ensuite, nous devons créer notre routine d'affichage. Encore une fois, c'est assez simple. La seule chose que vous pourriez ne pas reconnaître, c'est le « if e.has_key('keynamehere') ». C'est une vérification pour s'assurer qu'il y a effectivement des données dans les variables « Evaluation » et «Resume ». Certaines émissions n'ont pas cette information, aussi nous incluons la vérification pour améliorer les données que nous afficherons à l'écran (ci-dessus à droite).

Tout ce qui reste à faire est de mettre à jour notre routine « main » (en haut à droite de la page suivante). Encore une fois, je vais donner la routine « main » complète avec le nouveau code en caractères gras et en noir.

Maintenant, si vous enregistrez et exécutez le programme, la sortie de «_TrouveListeEpisodes » et « AfficheListeEpisodes » va fonctionner. En bas à droite se trouve un extrait de l'information d'un épisode.

C'est tout pour ce mois-ci. Comme toujours, vous pouvez trouver le code source complet sur pastebin : http://pastebin.com/gU5XSPcq. J'espère que jouer avec la bibliothèque vous amuse.

```
    def AfficheListeEpisodes(self,NomSerie,NombreSaisons,ListeEpisodes):
        print "-------------------------------------"
        print "Nom de la serie : %s" % NomSerie
        print "Nombre total de saisons : %s" % NombreSaisons
        print "Nombre total episodes : %d" % len(ListeEpisodes)
        print "-------------------------------------"
        for e in ListeEpisodes:
            print "Saison : %s" % e['Saison']
            print "   Numero Episode Saison : %s - Numero Episode Serie : %s" %
(e['NumeroEpisodeSaison'],e['NumeroEpisode'])
            print "   Titre: %s" % e['Titre']
            if e.has_key('Notation'):
                print "   Date Diffusion : %s   Notation : %s" %
(e['DateDiffusion'],e['Notation'])
            else:
                print "   Date Diffusion : %s   Notation : NONE" % e['DateDiffusion']
            if e.has_key('Resume'):
                print "   Resume : \n%s" % e['Resume']
            else:
                print "   Resume : NA"
            print "============================"
        print "------------ Fin de liste episodes ------------"
```

```
-------------------------------------
Series Name: Continuum
Total number of seasons: 2
Total number of episodes: 10
-------------------------------------
Season: 1
   Season Episode Number: 01 - Series Episode Number: 1
   Title: A Stitch in Time
   Airdate: 2012-05-27   Rating: 8.8
   Summary:
L'inspecteur Kiera Cameron perd tout ce qu'elle a et se retrouve avec une nouvelle mission quand
elle et huit terroristes dangereux sont transportés de leur époque, 2077, à 2012, pendant la
tentative des terroristes d'échapper à leur exécution. Elle prend une nouvelle identité et
rejoint le VPD [Ndt : Vancouver Police Department] afin d'arrêter le règne de violence des
terroristes. Sur le chemin, elle se lie d'amitié avec Alec Sadler, le jeune de 17 ans qui, un
jour, réussira à créer la technologie sur laquelle son monde à elle est construit.
============================
```

Il existe des données supplémentaires disponibles avec l'API que vous pouvez utiliser. S'il vous plaît rappelez-vous que TVRage fournit cette information gratuitement, alors pensez à leur faire un don pour aider leurs efforts dans la mise à jour de l'API et en guise de remerciements pour leur travail acharné.

```
def main():
    tr = TvRage()
    #--------------------
    # Chercher une serie par son nom
    #--------------------
    nom = raw_input("Entrer le nom de la serie -> ")
    if nom != None:
        liste = tr.TrouverIdParNom(nom)
        choix = tr.AfficheResultatsEmission(liste)
        if int(choix) == 0:
            sys.exit()
        else:
            option = int(choix)-1
            id = liste[option]['ID']
            print "Le ShowID choisi est %s" % id
    #-----------------------------
    # Recupere les informations
    #-----------------------------
    infoemission = tr.TrouveInfoEmission(id)
    #-----------------------------
    # Affiche les informations
    #-----------------------------
    tr.AfficheInfoEmission(infoemission)
    #-----------------------------
    # recupere la liste des episodes
    #-----------------------------
    NomSerie,SaisonsTotal,listeepisodes =
tr.TrouveListeEpisodes(id)
    #-----------------------------
    # Affiche la liste des episodes
    #-----------------------------
    tr.AfficheListeEpisodes(NomSerie,SaisonsTotal,listeepisodes)
    #-----------------------------
```

À la prochaine fois. Amusez-vous bien.

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.

Programmer en Python - Partie 42

Greg Walters

Supposons que vous avez décidé de créer un centre multimédia pour votre salle de séjour. Vous avez un ordinateur dédié à l'excellent programme XBMC. Vous avez passé des jours à ripper vos DVD de films et séries TV sur l'ordinateur. Vous avez fait la recherche et nommé les fichiers correctement. Mais disons que l'une de vos séries préférées est « NCIS », et que vous avez tous les épisodes possibles sur DVD. Vous avez aussi trouvé un endroit qui propose les épisodes actuels. Vous voulez savoir quel sera le prochain épisode et quand il sera diffusé. De plus, vous souhaitez créer une liste de tous les épisodes de séries TV que vous avez pour épater vos amis.

C'est le projet que nous allons commencer ce mois-ci. Notre première tâche consiste à fouiller dans le dossier contenant vos émissions de télévision, en récupérant le nom de la série et chaque épisode – y compris le nom et le numéro de la saison et le numéro de l'épisode. Toutes ces informations iront dans une base de données pour faciliter le stockage.

D'après XBMC, vous devriez nommer vos fichiers comme ceci pour chaque épisode :

```
Tv.Série.Nom.SxxExx.Nom de
l'épisode ici si vous
voulez.extension
```

Utilisons donc le tout premier épisode de NCIS (en VO) à titre d'exemple. Le nom de fichier pour un fichier AVI serait :

```
NCIS.S01E01.Yankee White.avi
```

et le tout dernier épisode serait :

```
NCIS.S10E17.Prime Suspect.avi
```

Si un nom d'émission contient plusieurs mots, il pourrait ressembler à ceci :

```
Doctor.Who.2005.S07E04.The
Power of Three.mp4
```

La structure du répertoire devrait ressembler à ceci :

```
émissions
2 Broke Girls
saison 1
Episode 1
Episode 2
...
saison 2
...
Doctor Who 2005
saison 1
...
saison 2
...
```

et ainsi de suite. Maintenant que nous savons ce que nous allons chercher et où ça se trouve, nous pouvons continuer.

Il y a très longtemps, nous avons créé un programme pour constituer une base de données contenant nos fichiers MP3. C'était dans le n° 35, je crois, au neuvième épisode de cette série. Nous avons utilisé une routine appelée ParcourirChemin pour entrer récursivement dans tous les dossiers à partir d'un chemin de départ, et récupérer les noms de fichiers avec l'extension « .mp3 ». Nous allons réutiliser une grande partie de cette routine et la modifier pour nos besoins. Dans cette version, nous rechercherons des fichiers vidéo qui ont une des extensions suivantes :

```
.avi
.mkv
.m4v
.mp4
```

Ce sont des extensions très courantes pour les fichiers vidéo dans le monde des médias PC.

Nous allons maintenant commencer avec la première partie de notre projet. Créez un fichier appelé « cherche_fichiers_tv.py ». Veillez à bien l'enregistrer quand nous aurons fini ce mois-ci, parce que nous allons repartir de là le mois prochain.

Commençons avec nos importations :

```
import os
from os.path import join,
getsize, exists
import sys
import apsw
import re
```

Comme vous pouvez le voir, nous importons les bibliothèques os, sys et apsw. Nous les avons toutes déjà utilisées. Nous importons aussi la bibliothèque re pour le support des expressions régulières. Nous allons en parler rapidement cette fois-ci, nous approfondirons dans le prochain article.

Maintenant, nous allons continuer avec nos deux dernières routines (page suivante). Tout le reste de notre code se trouvera entre les importations et ces deux dernières routines.

Voici (page suivante, en haut à droite) notre routine de travail principale. Nous y créons une connexion à la base de données SQLite fournie par apsw. Ensuite, nous créons un curseur pour interagir avec elle. Ensuite, nous appelons la routine FabriquerBase qui

va créer la base de données si elle n'existe pas.

Mes fichiers TV se trouvent sur deux disques durs. J'ai donc créé une liste pour contenir les chemins. Si vous avez un seul endroit, vous pouvez modifier les trois lignes comme suit :

```
dossierDepart =
"/chemin/dossier/"


ParcourirChemin(dossierDepart)
```

Ensuite, nous créons notre routine ifname « standard » :

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

Maintenant, tous les trucs ennuyeux sont faits, et nous pouvons passer au plat de résistance de notre projet. Nous allons commencer avec la routine FabriquerBase (au milieu à droite). Placez-la juste après les importations.

Nous avons déjà discuté de cette routine lorsque nous avons traité le scanner MP3, donc je vais juste vous rappeler que, dans cette routine, nous vérifions pour voir si la table existe et, sinon, nous la créons.

Maintenant, nous allons créer la routine ParcourirChemin (à droite, deuxième à partir du bas).

Lorsque nous entrons dans la routine (comme nous l'avons expliqué à l'époque), nous indiquons le chemin que nous allons parcourir. Nous vidons la variable nomEmission, que nous utiliserons plus tard, et ouvrons un fichier de log d'erreur. Ensuite, nous laissons la routine faire son boulot. Nous récupérons de l'appel (os.walk) un triplet (chemin du répertoire, noms de répertoires, noms de fichiers). Chemin du répertoire est une chaîne contenant le chemin vers le répertoire, noms de répertoires est une liste des noms des sous-répertoires dans le chemin, et noms de fichiers est une liste de noms des non-répertoires. Nous analysons ensuite la liste des noms de fichiers, pour vérifier si le nom se termine par une de nos extensions cibles.

```
#=======================================
def main():
    global connection
    global curseur

    # on cree la connexion et le curseur
    connection = apsw.Connection("EmissionsTV.db3")
    curseur = connection.cursor()
    FabriquerBase()
```

```
#=======================================
def FabriquerBase():
    # SI la table n'existe pas, on la cree
    # Sinon, on ignore ceci grace a la clause IF NOT EXISTS
    sql = 'CREATE TABLE IF NOT EXISTS EmissionsTV (pkID INTEGER PRIMARY KEY, Serie TEXT,
CheminRacine TEXT, NomFichier TEXT, Saison TEXT, Episode TEXT);'
    curseur.execute(sql)
```

```
#=======================================
def ParcourirChemin(chemin):

    nomEmission = ""
    # ouvre le fichier de log pour les erreurs
    ficerr = open('erreurs.log',"w")
    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'))]:
```

Maintenant, nous découpons le nom de fichier en séparant l'extension et le

```
    #=======================================
    # regle les chemins vers vos fichiers video
    #=======================================
    dossierDepart = ["/extramedia/tv_files","/media/freeagnt/tv_files_2"]
    for cptr in range(0,2):
        ParcourirChemin(dossierDepart[cptr])
    # ferme le curseur et la base de donnees
    curseur.close()
    connection.close()
    print("Fin")
```

nom du fichier (sans l'extension). Ensuite, nous appelons la routine RecupereSaisonEpisode pour avoir l'information de saison/épisode qui se trouve dans le nom du fichier, en supposant qu'il est correctement formaté.

```
NomFicOriginal,ext =
os.path.splitext(fic)

fl = fic

estok,donnees =
RecupereSaisonEpisode(fl)
```

RecupereSaisonEpisode retourne un booléen et une liste (dans ce cas « donnees ») qui contient le nom de la série, la saison et les numéros d'épisodes. Si un nom de fichier n'a pas le bon format, la variable booléenne « estok » (en haut à droite) sera fausse.

Ensuite (au milieu à droite), nous vérifions si le fichier est dans la base de données. Si c'est le cas, il ne faut pas le dupliquer. Nous vérifions simplement le nom du fichier. Nous pourrions aller plus loin et vérifier que le chemin est aussi le même, mais pour cette fois, c'est assez.

Si tout fonctionne correctement, la réponse de la requête ne devrait être que 1 ou 0. Si c'est 0, alors il n'est pas présent et nous allons écrire l'information dans la base de données. Sinon, nous passons à la suite. Remarquez la

```
            if estok:
                nomEmission = donnees[0]
                saison = donnees[1]
                episode = donnees[2]
                print("Saison {0} Episode {1}".format(saison,episode))
            else:
                print("Pas de Saison/Episode")
                ficerr.writelines('-------------------------\n')
                ficerr.writelines('{0} ne contient aucune information de
serie/episode\n'.format(fic))
                ficerr.writelines('-------------------------\n\n')
```

```
            requetesql = 'SELECT count(pkid) as nbLignes from EmissionsTV where NomFichier
= "%s";' % fl
            print(requetesql)
            try:
                for x in curseur.execute(requetesql):
                    nombreLignes = x[0]
                if nombreLignes == 0:  # vide, donc on ajoute
```

```
                    try:
                        sql = 'INSERT INTO EmissionsTV
(Serie,CheminRacine,NomFichier,Saison,Episode) VALUES (?,?,?,?,?)'
                        curseur.execute(sql,(nomEmission,racine,fl,saison,episode))
                    except:
                        print("Erreur")
                        ficerr.writelines('-------------------------\n')
                        ficerr.writelines('Erreur ecriture dans la base...\n')
                        ficerr.writelines('nomfic = {0}\n'.format(fic))
                        ficerr.writelines('-------------------------\n\n')
            except:
                print("Erreur")
            print('Serie - {0} Fichier - {1}'.format(nomEmission,fic))
```

commande try/except au-dessus et en-dessous. Si quelque chose va mal, comme un caractère que la base n'aime pas, cela empêchera le programme de s'arrêter. Cependant, nous enregistrerons l'erreur afin de pouvoir vérifier plus tard.

Nous insérons simplement un nouvel enregistrement dans la base de données ou écrivons dans le fichier d'erreur.

```
                # ferme le
fichier de log
ficerr.close
    # Fin de ParcourirChemin
```

Maintenant, regardons la routine RecupereSaisonEpisode.

```
#============================
============
def
RecupereSaisonEpisode(nomfic):
  nomfic = nomfic.upper()
  resp =
re.search(r'(.*).S\d\dE\d\d(\
.*)', nomfic, re.M|re.I)
```

La partie re.search du code vient de la bibliothèque re. Elle utilise un modèle

de chaîne et, dans ce cas, le nom du fichier que l'on veut analyser. re.M|re.I sont des paramètres qui disent que nous voulons utiliser une recherche de type multiligne (re.M) indépendante de la casse (re.I). Comme je l'ai dit précédemment, nous parlerons plus des expressions régulières le mois prochain, car notre routine correspondra à un seul type de chaîne de série/épisode. En ce qui concerne le modèle de recherche, nous recherchons : « .S » suivi de deux chiffres, suivis par « E » puis deux autres chiffres, puis un point. Si notre nom de fichier ressemblait à « tvshow.S01E03.avi », cela correspondrait. Cependant, certaines personnes codent leurs émissions ainsi : «tvshow.s01e03.avi », ou « tvshow.103.avi », ce qui rend la recherche plus difficile. Nous allons modifier cette routine le mois prochain pour couvrir la majorité des cas. Le « r' » permet qu'une chaîne brute soit utilisée pour la recherche.

Ensuite, la recherche retourne un objet correspondant que nous pouvons regarder. « rep » est une réponse qui est vide si aucune correspondance n'est trouvée, et, dans ce cas, deux morceaux d'information retournés. Le premier va nous donner les caractères jusqu'à la chaîne recherchée, et le second contiendra cette chaîne. Ainsi, dans le cas ci-dessus, group(1) serait « tvshow » et le second groupe serait « tvshow.S01E03. ».

Ceci est spécifié par les parenthèses de la recherche « (.*) » et « (\.*) ».

```
  si rep :
      nomEmission =
rep.group(1)
```

Nous récupérons le nom de l'émission dans le premier groupe. Puis nous calculons sa longueur de façon à pouvoir récupérer la série et l'épisode avec une commande de sous-chaîne.

```
      longueurNomEmission =
len(nomEmission) + 1
      se =
nomfic[longueurNomEmission:lo
ngueurNomEmission+6]
      saison = se[1:3]
      episode = se[4:6]
```

Ensuite, nous remplaçons tous les points de nomEmission par une espace, pour les rendre plus « lisibles par l'utilisateur ».

```
        nomEmission =
nomEmission.replace(".","  ")
```

Nous créons une liste contenant le nom de l'émission, la saison et l'épisode, et la retournons avec le booléen True pour dire que les choses se sont bien passées.

```
      ret =
[nomEmission,saison,episode]
      return True,ret
```

Sinon, si nous n'avons pas trouvé de correspondance, nous créons notre liste avec aucun nom de spectacle et deux « -1 », et la renvoyons avec un booléen False.

```
  else:
      ret = ["",-1,-1]
      return False,ret
```

Voilà tout le code. Maintenant, regardons à quoi le résultat devrait ressembler. En supposant que votre structure de fichier est exactement comme la mienne, une partie de l'affichage devrait ressembler à ceci :

```
Saison 02 Episode 04
SELECT count(pkid) as
nbLignes from EmissionsTV
where NomFichier =
"InSecurity.S02E04.avi";
Serie - INSECURITY Fichier -
InSecurity.S02E04.avi
Saison 01 Episode 08
SELECT count(pkid) as
nbLignes from EmissionsTV
where NomFichier =
"Prime.Suspect.US.S01E08.Unde
rwater.avi";
Serie - PRIME SUSPECT US
Fichier -
Prime.Suspect.US.S01E08.Under
water.avi
```

et ainsi de suite. Vous pouvez raccourcir la sortie si vous voulez pour éviter que l'écran ne vous rende fou. Comme nous le disions plus haut, chaque élément que nous trouvons sera placé dans la base de données. Quelque chose comme ceci :

```
pkID | Serie | Chemin Racine
| Nom du fichier | Saison |
Episode
2526 | NCIS |
/extramedia/tv_files/NCIS/Sea
son
7|NCIS.S07E04.Good.Cop.Bad.Co
p.avi | 7          | 4
```

Comme toujours, l'intégralité du code est disponible sur PasteBin.com à http://pastebin.com/p25nwCZM

La prochaine fois, nous traiterons un peu plus les formats de saison/épisode et ferons d'autres choses pour étoffer notre programme.

À 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.

TUTORIEL

Greg Walters

Programmer en Python - Partie 43

La dernière fois, nous avons commencé un projet qui finira par utiliser le module de TvRage que nous avons créé le mois d'avant. Nous allons maintenant poursuivre ce projet. Cette fois-ci, nous allons ajouter des fonctionnalités à notre programme : peaufiner la routine d'analyse de nom de fichier et ajouter deux champs (TvRageId et Etat) à la base de données. C'est parti !

Tout d'abord, nous modifierons nos lignes d'importation. Pour ceux qui viennent de nous rejoindre, je vais inclure celles de la dernière fois (en haut à droite).

Les lignes après « import re » sont nouvelles.

La chose suivante que nous allons faire est de réécrire la routine RecupereSaisonEpisode. Nous allons jeter à peu près tout ce que nous avons fait le mois dernier et le rendre plus souple à travers des schémas possibles de saisons/épisodes. Dans cette itération, nous serons en mesure de soutenir les schémas suivants :

```
Series.S00E00

Series.s00e00

Series.S00E00.S00E01

Series.00x00

Series.S0000

Series.0x00
```

Nous allons également corriger les problèmes éventuels de « zéro initial manquant » avant d'écrire dans la base de données.

Notre premier motif essaie d'attraper les fichiers multi-épisodes. Il existe différents systèmes de nommage, mais celui que nous prenons en charge ressemble à « S01E03.S01E04 ». Nous utilisons le modèle de chaîne « (.*)\.s(\d{1,2})e(\d{1,2})\.s(\d{1,2})e(\d{1,2}) ». Cela retourne (espérons-le) cinq groupes qui sont : le nom de la série (S[1]), la saison (S[2]), le numéro du premier épisode (S[3]), la saison (S[4]), et le numéro du deuxième épisode (S[5]). Rappelez-vous que les parenthèses créent les groupes retournés. Dans le cas ci-dessus, nous regroupons tout à partir du premier caractère jusqu'au « .s », puis deux chiffres, on passe le « e », puis deux chiffres, puis on recommence. Ainsi, le nom de fichier

```
import os
from os.path import join, getsize, exists
import sys
import apsw
import re
#----------------------------
#   NEW LINES START HERE
#----------------------------
from xml.etree import ElementTree as ET
import urllib
import string
from TvRage import TvRage
```

"Monk.S01E05.S01E06.avi" renvoie les groupes suivants :

```
S[1] = Monk

S[2] = 01

S[3] = 05

S[4] = 01

S[5] = 06
```

Nous utilisons uniquement des groupes S[1], S[2] et S[3] dans ce code, mais vous pouvez comprendre nos objectifs. Si nous trouvons une correspondance, nous réglons une variable nommée "Continuer" à Vrai. Cela nous permet de savoir ce que nous devrions faire après être passés à travers les différentes lignes If.

Ainsi, sur la page suivante (en haut à droite) se trouve le code de la routine RecupereSaisonEpisode.

Quand nous arrivons à ce point (page suivante, en bas à gauche) nous préparons le nom de l'émission en supprimant tous les points dans le nom de la série, puis extrayons les informations de la saison et de l'épisode des différents groupes, et les retournons. Pour l'information de saison, si nous avons un modèle comme « S00E00 », le numéro de saison aura un zéro. Mais si le modèle ressemble à « xxx », alors la saison est supposée être le premier caractère, et les deux suivants sont pour l'épisode. Afin d'être prévoyants, nous voulons que la saison soit un nombre à deux chiffres avec un zéro au début si nécessaire.

Ensuite, dans notre routine FabriquerBase, nous allons modifier l'instruction de création SQL pour ajouter les deux nouveaux champs (page suivante, en haut).

Encore une fois, la seule chose qui a changé depuis la dernière fois, ce sont les deux dernières définitions de champs.

Dans notre routine ParcourirChemin, les seuls changements sont les lignes qui sont réellement insérées dans la base de données, ceci afin de supporter la nouvelle structure. Si vous vous souvenez de la dernière fois, nous passons le dossier qui contient les fichiers TV à cette routine. Dans mon cas, il y a deux dossiers, ils sont donc placés dans une liste et nous utilisons une boucle pour passer chacun à la routine. En cours de la routine, nous parcourons chaque

```
def RecupereSaisonEpisode(nomfic):
    Continuer = False
    nomfic = nomfic.upper()
```

Ceci est notre première vérification de modèle.

```
    # devrait trouver des noms de fichiers de type multi-episodes .S01E01.S01E02
    rep = re.search(r'(.*)\.s(\d{1,2})e(\d{1,2})\.s(\d{1,2})e(\d{1,2})',nomfic, re.I)
    if rep:
        nomEmission = rep.group(1)
        Continuer = True
    else:
```

Notre deuxième vérification de modèle ressemble à SddEdd ou sddedd ...

```
        # devrait trouver SddEdd ou sddedd
        rep = re.search(r'(.*).S(\d\d?)E(\d\d?)(\.*)', nomfic, re.I)
        if rep:
            nomEmission = rep.group(1)
            Continuer = True
        else:
```

Le modèle suivant est pour ddxdd.

```
            # cherche ddxdd
            #rep = re.search(r'(.*).(\d\d?)x(\d\d?)(\.*)', nomfic, re.I)
            rep = re.search(r'(.*)\.(\d{1,2})x(\d{1,2})(.*)', nomfic, re.I)
            if rep:
                nomEmission = rep.group(1)
                Continuer = True
            else:
```

Ce modèle vérifie sdddd.

```
                # cherche Sdddd
                rep = re.search(r'(.*).S(\d\d)(.\d\d?)' , nomfic, re.I)
                if rep:
                    nomEmission = rep.group(1)
                    Continuer = True
                else:
```

Et enfin, nous essayons DDD

```
                # devrait trouver xxx
                    rep = re.search(r'(.*)(\d)(.\d\d?)',nomfic,re.I)
                    if rep:
                        nomEmission = rep.group(1)
                        Continuer = True
```

```
    if Continuer:
        longueurNomEmission =
len(nomEmission) + 1
        nomEmission =
nomEmission.replace("."," ")
        saison = rep.group(2)
        if len(saison) == 1:
            saison = "0" + saison
        episode = rep.group(3)
        ret = [nomEmission,saison,episode]
        return True,ret
    else:
        ret = ["",-1,-1]
        return False,ret
```

```
def FabriquerBase():
    # SI la table n'existe pas, on la cree
    # Sinon, on ignore ceci grace a la clause IF NOT EXISTS
    sql = 'CREATE TABLE IF NOT EXISTS EmissionsTV (pkID INTEGER PRIMARY KEY, Serie TEXT, CheminRacine TEXT, NomFichier
TEXT, Saison TEXT, Episode TEXT, tvrageid TEXT, etat TEXT);'
    curseur.execute(sql)
```

répertoire à la recherche de fichiers avec des extensions .avi, .mkv, .mp4 et .m4v. Lorsque nous trouvons un fichier qui correspond, nous l'envoyons à la routine RecupereSaisonEpisode. Nous vérifions ensuite si nous l'avons déjà entré dans la base de données et, sinon, nous l'ajoutons. Je vais vous donner (premier à droite) seulement une partie de la routine du mois dernier.

```
            sqlquery = 'SELECT count(pkid) as rowcount from TvShows where Filename =
"%s";' % fl
            try:
                for x in cursor.execute(sqlquery):
                    rcntr = x[0]
                if rcntr == 0:  # It's not there, so add it
                    try:
                        sql = 'INSERT INTO TvShows
(Series,RootPath,Filename,Season,Episode,tvrageid) VALUES (?,?,?,?,?,?)'
                        cursor.execute(sql,(showname,root,fl,season,episode,-1))
                    except:
```

Les deux lignes en noir sont nouvelles.

```
def ParcourirBase():
    tr = TvRage()
    SeriesCursor = connection.cursor()
    sqlstring = "SELECT DISTINCT series FROM TvShows WHERE tvrageid = -1"
```

Nous en sommes déjà à mi-chemin. Suivent quelques routines de support qui fonctionnent avec notre routine TvRage pour remplir les champs de la base. Notre première routine s'exécute après la routine ParcourirChemin, et parcourt la base de données pour obtenir le nom de la série et interroger le serveur de TvRage pour obtenir le numéro d'identification. Une fois que nous avons cela, nous mettons à jour la base de données, puis utilisons à nouveau le numéro d'identification sur TvRage pour obtenir l'état actuel de la série. Cet état peut être « New Series », « Returning Series », « Canceled », « Ended » et « On Haitus » (série nouvelle, de retour, annulée, terminée, en pause). La raison pour laquelle nous voulons cette information est que, lorsque nous allons vérifier les nouveaux épisodes, nous ne voulons pas nous embêter avec des séries qui n'auraient pas de nouveaux épisodes parce qu'elles sont annulées. Ainsi, nous en avons maintenant l'état et pouvons l'écrire dans la base de données (ci-dessus).

Nous allons nous arrêter ici dans notre code pendant un instant pour regarder la requête SQL que nous utilisons. C'est un peu différent de tout ce que nous avons fait auparavant. La chaîne est :

```
SELECT DISTINCT series FROM
TvShows WHERE tvrageid = -1
```

Ce qui dit : donne-moi un seul exemple du nom de la série, peu importe combien il y en a, où le champ tvrageid vaut « -1 ». Si, par exemple, nous avons 103 épisodes de Doctor Who 2005, en utilisant le Distinct, je vais recevoir un seul enregistrement, en supposant que nous n'avons pas encore obtenu un TvRageID.

```
  for x in
CurseurSerie.execute(requetes
ql):

      nomSerie = x[0]

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

Nous utilisons la routine capwords

```
def MettreAJourBase(nomSerie,id):
    idcurseur = connection.cursor()
    requetesql = 'UPDATE EmissionsTV SET tvrageid = ' + id + ' WHERE serie = "' + nomSerie + '"'
    try:
        idcurseur.execute(requetesql)
    except:
        print "error"
```

```
def RecupererEtatEmission(nomSerie,id):
    tr = TvRage()
    idcurseur = connection.cursor()
    dict = tr.TrouveInfoEmission(id)
    etat = dict['Etat']
    requetesql = 'UPDATE EmissionsTV SET Etat = "' + etat + '" WHERE serie = "' + nomSerie + '"'
    try:
        idcurseur.execute(requetesql)
    except:
        print "Erreur"
```

de la bibliothèque string pour changer le nom de la série (x[0]) « de façon correcte » puisque nous stockons les noms des émissions en majuscules. Nous faisons cela parce que TvRage s'attend à recevoir quelque chose d'autre que des seules majuscules et nous n'obtiendrons pas les résultats que nous recherchons. Ainsi, le nom de la série « THE MAN FROM UNCLE » sera converti en « The Man From Uncle ». Nous utilisons cela dans l'appel à la fonction TrouverIdParNom de notre bibliothèque TvRage. Cela récupère la liste des émissions correspondantes et les affiche pour qu'on choisisse la meilleure. Une fois que nous en avons choisi une, nous mettons à jour la base de données avec le numéro d'identification, puis appelons la routine RecupererEtatEmission pour obtenir le statut en cours depuis TvRage (en bas).

La routine MettreAJourBase (en haut) utilise simplement le nom de la série comme clé pour mettre à jour tous les enregistrements avec l'ID approprié de TvRage.

RecupererEtatEmission (ci-dessus) est également très simple. Nous appelons la routine TrouveInfoEmission de la bibliothèque TvRage en passant l'id que nous venons de récupérer de TvRage pour obtenir l'information sur la série. Si vous vous souvenez, TvRage fournit beaucoup d'informations sur

```
        print("Requesting information on " + searchname)
        sl = tr.FindIdByName(searchname)
        which = tr.DisplayShowResult(sl)
        if which == 0:
            print("Nothing found for %s" % seriesname)
        else:
            option = int(which)-1
            id = sl[option]['ID']
            UpdateDatabase(seriesname,id)
            GetShowStatus(seriesname,id)
```

```
dossierDepart = ["/extramedia/tv_files","/media/freeagnt/tv_files_2"]
    #for cptr in range(0,2):
        #ParcourirChemin(dossierDepart[cptr])
    ParcourirBase()
    # ferme le curseur et la base de donnees
    curseur.close()
    connection.close()

    print("Fin")
```

la série, mais tout ce qui nous intéresse à ce stade est l'état de l'émission. Puisque tout est retourné dans un dictionnaire, il nous suffit de chercher la clé ['Etat']. Une fois que nous l'avons, nous mettons à jour la base de données avec, puis passons à autre chose.

Nous avons presque terminé notre code. Nous ajoutons pour finir une ligne à notre routine principale du mois dernier (en noir bas de page précédente) pour appeler la routine ParcourirBase après avoir récupéré tous les noms de fichiers. Encore une fois, je vais vous donner seulement une partie de la routine Main, juste pour que vous puissiez trouver le bon endroit pour mettre la nouvelle ligne.

C'est terminé pour le code. Examinons mentalement ce qui arrive quand nous exécutons le programme.

Tout d'abord, nous créons la base de données si elle n'existe pas.

Ensuite, nous parcourons les chemins prédéfinis, à la recherche de fichiers qui ont une des extensions suivantes :

```
.AVI, .MKV, .M4V, .MP4
```

Lorsque nous en trouvons un, nous essayons d'analyser le nom du fichier à la recherche d'un nom de série, d'un numéro de saison et d'un numéro d'épisode. Nous prenons cette information et la mettons dans une base de données, si elle n'y existe pas déjà.

Après avoir recherché les fichiers, nous interrogeons la base de données à la recherche de noms de séries pour lesquelles il n'y a pas d'ID TvRage associé. Nous interrogeons alors l'API TvRage et demandons des fichiers correspondants pour trouver cet ID. Chaque série va passer par cette étape une fois. Le code encadré ci-après montre les options pour, dans ce cas, la série « Midsomer Murders ».

J'ai saisi (dans ce cas) 1, qui associe cette série avec l'ID TvRage 4466. Il est entré dans la base de données, et nous utilisons alors cet ID pour demander l'état actuel de la série, toujours sur TvRage. Dans ce cas, on nous renvoie « Returning Series ». L'état est alors entré dans la base de données et nous continuons.

Le passage initial dans la base de données prendra un certain temps et nécessitera votre attention, parce que chaque série doit poser des questions sur le numéro d'identification correspondant. La bonne nouvelle est que ceci n'est fait qu'une seule fois. Si vous êtes « normal », vous n'en aurez pas tant que ça à traiter. J'ai eu 157 séries différentes à passer et il a donc fallu un peu de temps. Comme j'ai été prudent lorsque j'ai saisi mes noms de fichiers (en vérifiant sur TvRage et TheTvDB.com pour avoir la formulation correcte du nom de la série), la majorité des réponses a été l'option n° 1.

```
Requesting information on Midsomer Murders
5 Found
-----------------------
1 - Midsomer Murders - 4466
2 - Motives and Murders - 31373
3 - See No Evil: The Moors Murders - 11199
4 - The Atlanta Child Murders - 26402
5 - Motives & Murders: Cracking the Case - 33322
Enter Selection or 0 to exit ->
```

Juste pour votre information, plus de la moitié des séries télévisées que j'ai sont terminées ou ont été annulées. Cela devrait vous donner une idée de mon âge approximatif.

Le code complet est, comme toujours, disponible sur Pastebin : http://pastebin.com/DgwmTMHr

[NdT : code traduit par l'équipe francophone. Pour le code original, voir http://pastebin.com/MeuGyKpX.]

La prochaine fois, nous continuerons l'intégration avec TvRage. D'ici là, passez un bon mois !

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.

python