SAÉ 203

DOCUMENTATION DU CODE « aggreg.py »

Le but de cette SAÉ était de développer et déployer une solution informatique complète en respectant un cahier des charges.

Pour cela j'ai dû utiliser un code python composé de plusieurs fonctions et c'est de ce code python que ce document traitera.

Conception et test du programme :

Alors pour concevoir ce programme j'ai tout d'abord choisi la manière dont j'allais programmer.

Plusieurs options s'offraient à moi mais j'ai décidé de programmer sur mon propre PC (j'ai utilisé Visual Studio Code), qui utilisaient tout d'abord des flux RSS provenant de sites internet mais à un certain moment du projet lorsqu'il était assez avancé j'ai commencé à utiliser des flux générés par mon serveur (simu-site). Lorsqu'il a fallu tester le programme sur mes serveurs je l'ai envoyé sur l'agrégateur pour vérifier qu'il fonctionnait bien puis à un moment donné, plus vers la fin, j'ai décidé de programmer directement sur la machine « aggreg » afin de limiter les erreurs lors des échanges de programme. (Avec « nano »)

Fonctionnement du programme :

Mon programme, comme indiqué précédemment, utilise plusieurs fonctions. Chacune de ces dernières représentent les étapes de l'agrégation des flux RSS. Nous allons détailler le code :

Avant toute chose les premières lignes du document sont les suivantes :

```
#!/usr/bin env python3
# -*- coding: utf-8 -*-
```

Ces lignes servent à indiquer la position de l'interpréteur python et rendent le programme exécutable sur le shell. L'encodage détermine les caractères qui vont être utilisés.

Ensuite j'ai utilisé plusieurs modules que j'ai importés, qui vont nous permettre d'utiliser certaines commandes pour réaliser l'agrégateur :

```
import feedparser → Manipuler les flux RSS
import time → Affichage du temps
import yaml → Manipuler les fichiers YAML
```

Nous allons voir plus tard quand est-ce que nous en avons eu besoin dans le programme.

Pour commencer il y a la première fonction nommée « charge_urls ». Celle-ci prend en paramètre une liste d'URL et nous renvoie le flux RSS de chaque URL se trouvant dans la liste. Pour y parvenir, la fonction fait appel au module évoqué plus tôt, « feedparser ».

```
#décodage du flux

          feed = feedparser.parse(url)
```

Elle est aussi capable de faire la différence entre un lien existant et un autre inexistant grâce un élément du flux RSS appelé « bozo ». Lorsqu'un lien n'existe pas elle mettra dans sa liste de flux « None » pour le lien correspondant.

```
if feed["bozo"]== False :
          liste.append(feed)
     else :
          liste.append(None)
```

La prochaine fonction s'appelle « fusion_flux ». Les paramètres utilisés sont :

- La même liste d'URL utilisée pour la fonction « charge_urls »
- La liste renvoyée par la fonction « charge_urls »

Cette fonction a pour but de récupérer des informations précises, tels que le titre, le serveur de provenance, la date, etc…, sur chaque flux RSS contenus dans la liste renvoyée par « charge_urls ». Elle aussi traitera seulement les liens existants. La fonction renvoie son résultat sous la forme d'une liste qui contient des dictionnaires. Chaque dictionnaire rassemble les infos voulues pour chaque flux.

Une petite fonction nommée « yaml_conf » est utilisée seulement pour ouvrir le fichier YAML qui va contenir les informations des serveurs et des fichiers RSS. Elle ne prend aucun paramètre, elle va juste charger le fichier et rendre les données contenues utilisables grâce au module « yaml » évoqué au début.

La fonction qui suit, « genere_html », comme son nom l'indique permet de générer une page HTML qui va contenir le détail des flux récupérés. Pour cela elle a besoin de ce qu'a collecté la fonction « charge_urls », c'est pour ça qu'elle prend en paramètre :

- La liste renvoyée par « charge_urls »
- Un chemin lui indiquant où générer cette page HTML

Sur cette page on pourra y retrouver les différents flux ainsi que leurs informations. Pour créer la page, chaque ligne va écrire une ligne de la page HTML, il faut donc respecter la syntaxe pour ne pas avoir d'erreur. Dans cette fonction, on utilise le module « time ». Il va nous permettre d'afficher l'heure et la date sur la page afin d'indiquer à quelle heure la page a été actualisée.

Voici la ligne qui utilise le module, elle utilise une syntaxe très spéciale :

```
page.write("\t\t\t<p>"+ time.strftime("%a, %d %b %Y %H:%M:%S")+"</p> \n")
```

La fonction contient aussi l'import de la feuille de style CSS qui permet de personnaliser l'affichage et le style de la page en modifiante le fichier « style.css »

Pour terminer sur la conception du code nous allons parler du « main ». Dedans on peut y retrouver plusieurs choses.

Tout d'abord on fait appel à la fonction « yaml_conf » et on la stocke dans une variable. Etant donné que cette fonction doit ouvrir le fichier « conf.yaml » de l'agrégateur, qui lui va contenir des paramètres qui vont nous servir (url des serveurs, chemin du fichier html, nom des fichiers RSS) sous la forme de dictionnaire, on va stocker dans des variables les valeurs des différents paramètres.

Ensuite on va appeler les fonctions dans des variables avec les paramètres appliqués (ceux stockés précédemment le cas échéant) afin d'utiliser seulement des mots simples lorsque l'on voudra faire appel à toutes les fonctions.

Dans le « main » se trouve aussi une petite boucle qui va permettre d'ajouter le nom du fichier RSS aux urls des serveurs.

Et enfin pour finir on va appeler la fonction « genere_html » avec tous ses paramètres pour générer la page.

Limites et améliorations :

On peut voir que l'un des points faibles de notre programme est que le fichier de configuration doit être de la même syntaxe que l'on a choisie il, pourrait s'adapter.

Il y a aussi le codage de la page HTML, on n'a pas opté pour une solution très optimale mais elle fonctionne quand on ne sait pas faire autrement. Il faudrait peut-être changer cela pour simplifier le code.

Pour ce qui est des améliorations on pourrait tout d'abord modifier le CSS. Certes il est présent mais je pense qu'il pourrait être plus travaillé comme écrire les catégories en couleur en fonction de la gravité de l'évènement, ou alors activer le tri des flux par ordre chronologique. J'ai tenté de le faire en indiquant cela dans mon fichier de configuration :

```
tri-chrono: true
```

Mais ensuite je ne savais pas comment faire au niveau de python.

Conclusions :

On a pu voir comment le code python se comporte et les raisons des choix de certains modules, certaines méthode, etc…

Même si le programme fonctionne on a pu voir certaines limites du programme et donc apporter des solutions, on peut voir qu'il y a quand même des modifications à apporter afin de mieux comprendre et lire les flux.