Explication du fonctionnement du programme aggreg.py

Récupérations des sources (serveur) :

recupflux va récupérer les flux à traiter grâce à un fichier flux.yaml remplis au préalable.

```
def recupflux(fichieryaml):
    #j'ouvre le fichier en lecture
    with open(fichieryaml, 'r') as f:
        f=yaml.safe_load(f)
        #je récupère ce dont j'ai besoin (url,nom du fichier,chemin html)
        source= f["sources"]
        files=f["rss-name"]
        html=f["destination"]
        chrono=f["tri-chrono"]
        #liste que je vais renvoyer
        listef=[]
        #je fait une boucle for pour mettre le tout dans ma listef
        for i in range(len(source)):
            t=source[i]
            #je combine l'url avec l'extension du fichier
            t+=files
            #je l'ajoute a listef
            listef.append(t)
        #je retourne listef et html
        return(listef,html, chrono)
```

Pour pouvoir ouvrir le fichier et lire son contenu j'ai utilisé le module yaml :

```
import yaml
```

Ainsi que ces deux commandes pour le charger et lire son contenu :

```
with open(fichieryaml, 'r') as f:
    f=yaml.safe_load(f)
```

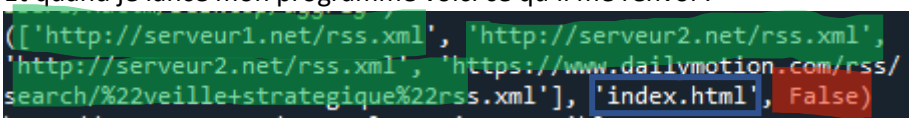
Je fais une boucle for pour récupérer l'url de chaque site et leurs ajouter l'extension du fichier puis j'ajoute le tout dans ma listef :

```
for i in range(len(source)):
    t=source[i]
    #je combine l'url avec l'extension du fichier
    t+=files
    #je l'ajoute a listef
    listef.append(t)
```

Voici un exemple de fichier yaml :

```
sources:
  - http://serveur1.net/
  - http://serveur2.net/
  - http://serveur3.net/
rss-name: rss.xml
destination: /var/www/aggreg/index.html
tri-chrono: True
```

Et quand je lance mon programme voici ce qu'il me renvoi :



Pour la première variable retourner par la fonction on a une liste dans laquelle on retrouve les url, puis dans la seconde on retrouve le chemin exact vers le fichier HTML et enfin le False du tri-chrono.

Récupération des flux rss :

Charges_urls va se charger de récupérer tout le contenu rss des sites :

```
def charges_urls(liste_url):
    #j'initialise la liste finale
    fluxRss=[]
    news_feed=[]
    #je créer un boucle for pour itérer dans la liste_url
    for rss in range(len(liste_url)):
        #je charge le document rss de la liste_url
        xml=feedparser.parse(liste_url[rss])
        news_feed.append(xml)
        #vérifie si le documents rss renvoi de l'information ou pas
        if news_feed[rss]["bozo"]==False:
            #si il renvoi rien on ajoute None a la liste
            new=news_feed[rss].get('entries')
            fluxRss.append(new)
        else:
            #sinon on ajoute le document rss a la liste
            fluxRss.append(None)
    #on renvoi la liste des flux
    return(fluxRss)
```

Pour cela je créer une boucle dans laquelle je vais ouvrir chaque fichier de l'url.
J'ai donc besoin du module feedparser qui permet la lecture des fichiers au format xml :

```
import feedparser
```

Avec ceci je peux donc lire les fichiers et les ajouter à une variable pour pouvoir les traités :

```
xml=feedparser.parse(liste_url[rss])
news_feed.append(xml)
```

Au lieu de bêtement ouvrir le contenu d'un flux rss et l'ajouter a ma variable finale, j'effectue une vérification pour savoir si le fichier renvoi bien un flux.

Pour cela j'utilise une clé dans le dictionnaire qui est bozo dont la valeur varie entre False et True.
Si bozo:False cela veut dire que nous avons bien un flux rss à l'intérieur du fichier, si c'est l'inverse cela veut dire qu'il n'y a pas de flux rss.

Voici comment j'ai effectué cette vérification en python :

```
#vérifie si le documents rss renvoi de l'information ou pas
if news_feed[rss]["bozo"]==False:
    #si il renvoi rien on ajoute None a la liste
    new=news_feed[rss].get('entries')
    fluxRss.append(new)
else:
    #sinon on ajoute le document rss a la liste
    fluxRss.append(None)
```

Une fois le programme lancer voici ce qu'il nous renvoi (j'ai laissé que le début car c'est trop grand) :

```
[None, None, None, [{'title': 'Le présentéisme, un phénomène très
français', 'title_detail': {'type': 'text/plain', 'language': None,
'base': 'https://www.dailymotion.com/rss/search/%22veille+strategique
%22rss.xml', 'value': 'Le présentéisme, un phénomène très français'},
'links': [{'rel': 'alternate', 'type': 'text/html', 'href': 'https://
www.dailymotion.com/video/x8bp4tf'}], 'link': 'https://
www.dailymotion.com/video/x8bp4tf', 'summary': 'Travailler plus pour
être valorisé plus…. cette culture semble très ancrée dans les esprits
français et espagnols.<br />Le présentéisme est le fait de rester sur
```

None sont les url qui ne sont pas accessible. Quant au flux qui a marché il s'affiche bel et bien.

Trie de l'information des flux :

fusion_flux va permettre de récupérer uniquement les information que l'on veut afficher et donc va trier les flux rss.

```
def fusion_flux(liste_url,liste_flux):
    #j'initialise la liste finale
    final=[]
    dico={}
    #je créer un boucle for pour itérer dans la liste_flux
    for i in range(len(liste_flux)):
        flux=liste_flux[i]
        #vérifie si le flux contient bien quelque chose
        if flux==None:
            print(liste_url[i],"est inaccessible ")
        else:
            #si le fichier est disponible je récupère ce que je veux
            for rss in range(len(flux)):
                doc=flux[rss]
                tags2=flux[rss].get('tags')
                dico['titre']=doc.get('title')
                dico['categorie']=tags2[0].get('term')
                serveur=doc.get("link")
                serveur=serveur[7:15]
                dico['serveur']=str(serveur)
                dico['date_publi']=doc.get('published')
                dico['lien']=doc.get('link')
                dico['description']=doc.get('summary')
                dico['guid']=doc.get('guid')
                final.append(dico)
                dico={}
    return(final)
```

Pour cela Il nous suffit juste de prendre un à un les flux et de les triés. Vu que l'on peut avoir des flux « None » j'ai fait une vérification pour afficher à l'utilisateur que ce site est indisponible :

```
#je créer un boucle for pour itérer dans la liste_flux
for i in range(len(liste_flux)):
    flux=liste_flux[i]
    #vérifie si le flux contient bien quelque chose
    if flux==None:
        print(liste_url[i],"est inaccessible ")
```

Et voici ce que ça donne :

```
In [8]: runfile('C:/Users/Hatim/Downloads/aggreg8.py', wdir='C:/Users/
Hatim/Downloads')
http://serveur1.net/rss.xml est inaccessible
http://serveur2.net/rss.xml est inaccessible
http://serveur2.net/rss.xml est inaccessible
```

Quand le flux marche on récupère ce dont on a besoin dans le dico grâce à la commande get :

```
else:
    for rss in range(len(flux)):
        doc=flux[rss]
        tags2=flux[rss].get('tags')
        dico['titre']=doc.get('title')
        dico['categorie']=tags2[0].get('term')
        serveur=doc.get("link")
        serveur=serveur[7:15]
        dico['serveur']=str(serveur)
        dico['date_publi']=doc.get('published')
        dico['lien']=doc.get('link')
        dico['description']=doc.get('summary')
        dico['guid']=doc.get('guid')
        final.append(dico)
        dico={}
```

Enfin on renvoi le résultat :

```
#on renvoi la liste des flux
return(fluxRss)
```

Génération de la page HTML :

genere_html va permettre de générer la page html qui sera accessible depuis le client.

```
def genere_html(liste_evenements, chemin_html):
    #je récupère l'heure
    date = datetime.now()
    datef=str(date)
    #j'enlève les 19 caractère derrière
    datef=datef[:19]
    listef=[]
    #j'ouvre le fichier html en écriture
    with open (chemin_html, 'w+',encoding='utf-8') as files:
        #je prépare tout le header et le body
        html_head=['<!DOCTYPE html>','<html lang="en">','<head>',
                   '<meta charset="utf-8">',
                   '<meta name="viewport" content="width=device-width, initial-scale=1">',
                   '<title>Events log</title>',
                   '<link rel="stylesheet" href="style.css" type="text/css"/>',
                   '</head>']

        html_body=['<body>','<article>','<header>','<h1>Events log</h1>','</header>',
                   '<p class="heure">',datef,'</p>','</body>']

        #je fait une boucle for pour récupérer les informations
        for e in range(len(liste_evenements)-1):
            serveur=liste_evenements[e]['serveur']
            date_pub=liste_evenements[e]['date_publi']
            categorie=liste_evenements[e]['categorie']
            guid=liste_evenements[e]['guid']
            lien=liste_evenements[e]['lien']
            description=liste_evenements[e]['description']
            #ici j'attribu une class en fonction de l'erreur
            if categorie=="MINOR":
                categorie="<p class='minor'>"+categorie
            if categorie=="MAJOR":
                categorie="<p class='major'>"+categorie
            if categorie=="CRITICAL":
                categorie="<p class='critical'>"+categorie
            #je met tout dans une variable sous forme de liste
            article= [
                '<article>','<p class="serveur">','from: '+serveur,'</p>'
                '<p>','pubDate: '+date_pub,'</p>'
                '<p>'+categorie,'</p>'
                '<p>','guid: '+guid,'</p>'
                '<p><a href="{link}">','link: '+lien,'</a></p>'
                '<p>','description: '+description,'</p>','</article>'
                ]
            #j'ajoute à ma listef
            listef.append(article)

        #enfin j'écris dans le fichier
        files.writelines(html_head)
        files.writelines(html_body)
        for i in range(len(listef)):
            files.writelines(listef[i])
```

Pour récupérer la date et l'heure exacte j'utilise le module time :

```
from time import strftime, localtime, time
```

Puis je stock la date et l'heure dans une variable en la convertissant au format str car sinon elle est illisible ici a=jour en lettre, d=jour en chiffre, b=nom du mois, Y=année, H=heure,M=minutes,S=secondes et Z=zone géographique:

```
#je récupère l'heure
datef=strftime('%a, %d %b %Y %H:%M:%S %Z' ,localtime(time()))
```

J'ouvre le fichier en écriture, « w+ » signifie que si le fichier n'est pas créer il le sera :

```
#j'ouvre le fichier html en écriture
with open (chemin_html, 'w+',encoding='utf-8') as files:
```

Je prépare dans deux variable de type liste mon html pour mon head et mon body :

```
#je prépare tout le header et le body
html_head=['<!DOCTYPE html>','<html lang="en">','<head>',
           '<meta charset="utf-8">',
           '<meta name="viewport" content="width=device-width, initial-scale=1">',
           '<title>Events log</title>',
           '<link rel="stylesheet" href="style.css" type="text/css"/>',
           '</head>']

html_body=['<body>','<article>','<header>','<h1>Events log</h1>','</header>',
           '<p class="heure">',datef,'</p>','</body>']
```

Je prépare également mes informations de flux rss dans une liste pour cela je fait une boucle for ou je les récupères :

```
 #je fait une boucle for pour récupérer les informations
 for e in range(len(liste_evenements)-1):
     serveur=liste_evenements[e]['serveur']
     date_pub=liste_evenements[e]['date_publi']
     categorie=liste_evenements[e]['categorie']
     guid=liste_evenements[e]['guid']
     lien=liste_evenements[e]['lien']
     description=liste_evenements[e]['description']
```

Le -1 est mis car la dernière valeur est tout le temps vide et donc cela évite les erreurs

J'ai attribuer une class en fonction du type d'erreur, cela me permet de mettre la couleurs aproprié sur mon css :

```
#ici j'attribu une class en fonction de l'erreur
if categorie=="MINOR":
    categorie="<p class='minor'>"+categorie
if categorie=="MAJOR":
    categorie="<p class='major'>"+categorie
if categorie=="CRITICAL":
    categorie="<p class='critical'>"+categorie
```

Je place le tout dans une liste puis j'ajoute cette dernière à ma listef :

```
#je met tout dans une variable sous forme de liste
article= [
    '<article>','<p class="serveur">','from: '+serveur,'</p>'
    '<p>','pubDate: '+date_pub,'</p>'
    '<p>'+categorie,'</p>'
    '<p>','guid: '+guid,'</p>'
    '<p><a href="{link}">','link: '+lien,'</a></p>'
    '<p>','description: '+description,'</p>','</article>'
    ]
#j'ajoute à ma listef
listef.append(article)
```

Enfin j'écris dans l'ordre toutes les listes préparés juste avant :

```
#enfin j'écris dans le fichier
files.writelines(html_head)
files.writelines(html_body)
for i in range(len(listef)):
    files.writelines(listef[i])
```

Je peux également trié dans l'ordre chronologique grâce a la date de l'erreur :

```
#Génération HTML avec ou sans tri chronologique
if tri == True:
    liste_tri=sorted(fusion_flux(site,charges_urls(site)), key=lambda x:datetime.strptime(x["date_publi"], '%a, %d %b %Y %H:%M'),reverse=True)
    genere_html(liste_tri, html)

else:
    genere_html(fusion_flux(site,charges_urls(site)), html)
```

Les amélioration possible serait de rendre le proramme plus compacte et rapide.

Les limitations serait d'après moi le nombre de flux rss. Je pense qu'à un certain nombre de flux conséquent le programme crashera car la mémoire tampon sera pleine à craqué.