Groupe 4 : Robot

BENJEBRIA Sofian, BESNARD Léa, DELOEUVRE Noémie, HBABOU Nabil, HERVE Pierrick, KRUMEICH Laetitia, MOTHES Céline, PELAT Aurélien, SECK Mamadou Moustapha, SEGUELA Morgan

GROUPE 4
Projet inter-promotion
Année universitaire : 2017/2018

S O M M A I R E

1. Objectif du groupe

L'objectif de notre groupe est d'implémenter un robot qui récupère le contenu des articles de journaux en ligne. Le robot réalise donc les tâches suivantes :

- Accéder à la page principale d'un journal
- Se connecter avec un compte d'abonné [Optionnel]
- Parcourir l'arborescence du site du journal pour accéder aux articles
- Récupérer le contenu des articles (ceux qui n'ont pas déjà été aspirés)
- Récupérer des métadonnées utiles telles que les catégories thématiques, l'auteur ou les auteurs, la date de publication, le contenu, le titre et le nom du journal
- Stocker toutes les informations récupérées dans des fichiers JSON (un fichier par article)
- Transmettre les fichiers JSON au programme réalisé par le groupe 5 (filtrage)

2. Organisation

2.1. Organisation générale

Notre travail consistait surtout à la récupération d'articles de différents journaux, nous avons donc divisé le travail en deux groupes en partageant les L3 - M1 - M2 pour améliorer la compréhension générale du sujet et des solutions. Puis, nous nous sommes divisés en 4 groupes pour encore plus paralléliser le travail. Enfin, chaque personne a pris un journal pour avoir le maximum de journaux la première semaine.

Certains ont ensuite testé les codes dans le cadre du serveur afin de pouvoir les mettre sur le serveur.

Lorsque tous les journaux ont été extraits, nous avons commencé à travailler sur la deuxième version.

Nous avons utilisé la méthode de gestion de projet SCRUM qui s'inscrit dans le cadre des méthodes Agiles. Concrètement, nous avons fait une courte réunion chaque matin et soir pour récapituler le travail réalisé dans la journée, les éventuels problèmes rencontrés et ce qui pourrait les résoudre ainsi que le travail qui sera réalisé le lendemain.
Cette méthode de travail nous a permis de nous adapter rapidement aux changements de spécifications provenant de l'équipe de direction et des autres groupes (ex. ajout d'un attribut "thème" pour articles collectés, suppression des accents dans le contenu récupéré).

2.2. Diagrammes de GANTT

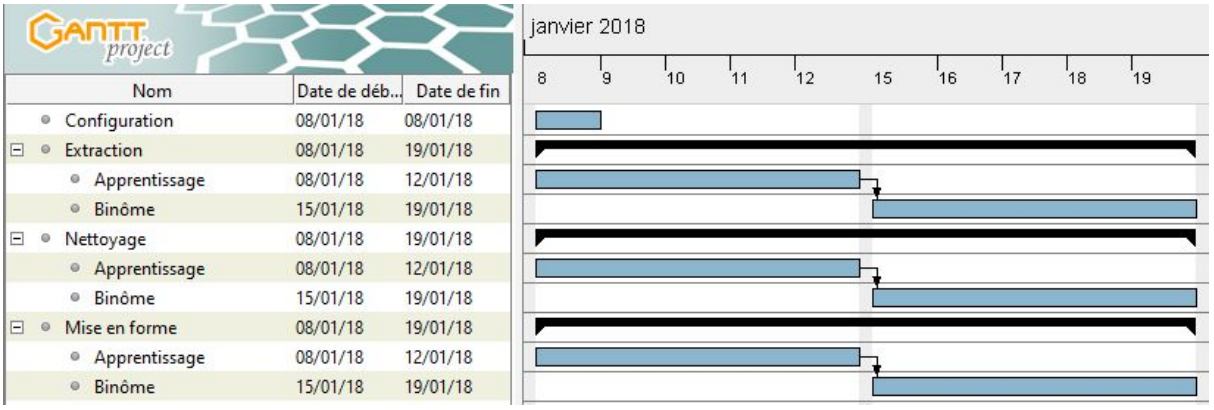


Diagramme de Gantt prévisionnel

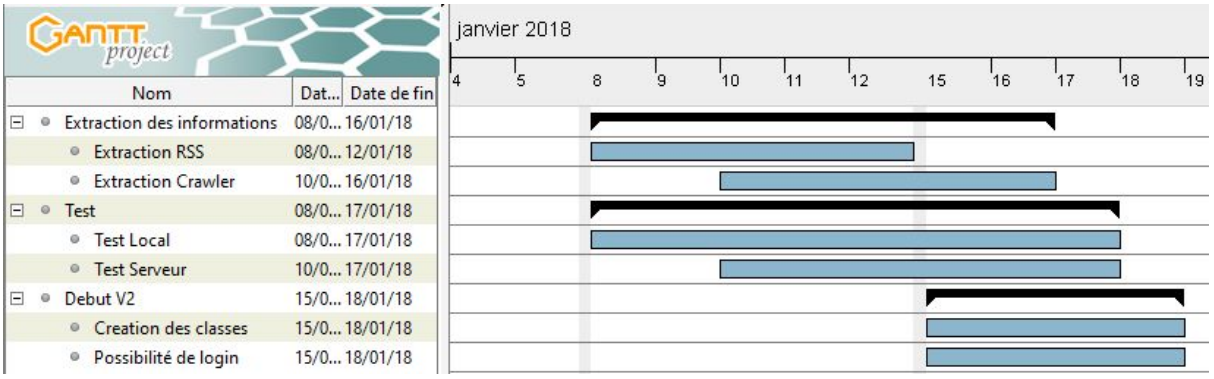


Diagramme de Gantt réel

Cette différence s'explique notamment par le fait qu'il a été difficile d'évaluer le temps d'apprentissage. De plus, certaines caractéristiques (notamment celles de la 2ème version) ont été réalisées par la suite.

2.3. Outils utilisés

Nous avons utilisé plusieurs outils pour organiser le travail au sein de l'équipe, ils sont les suivants :

- la messagerie Slack
- la plateforme GitHub basée sur Git
- les outils Git Kraken et GitHub Desktop : pour accéder facilement à GitHub et pouvoir se concentrer au maximum sur notre objectif plutôt que d'apprendre à utiliser Git en ligne de commande

Le rôle de responsable communication (coordination avec les autres groupes) a été assuré durant la première semaine par Pierrick HERVE puis pendant la deuxième semaine par Morgan SEGUELA.

2.4. Méthode qualité

Nous avons mis en place les contrôles de qualité suivants au sein du groupe :

- activation sur notre environnement de travail, des analyseurs de codes qui nous signalent lorsque nous violons une contrainte de la pep8
- la mise en place d'un fichier g4_utils qui contient toutes les fonctions communes pour tous les journaux. Ceci nous a permis donc de factoriser au mieux notre code.
- découpage de notre code en fonctions afin de faciliter sa lisibilité.
- à chaque nouvelle version du code réalisée par un membre du groupe, un autre membre exécute le code en local et lui fait part de ses éventuelles remarques
- Puis lorsqu'un ensemble d'outils fonctionne, la personne qui communique avec le serveur teste dans le contexte du serveur les outils. Si cela fonctionne, la personne le met sur le serveur.

3. Déroulement de la mission principale

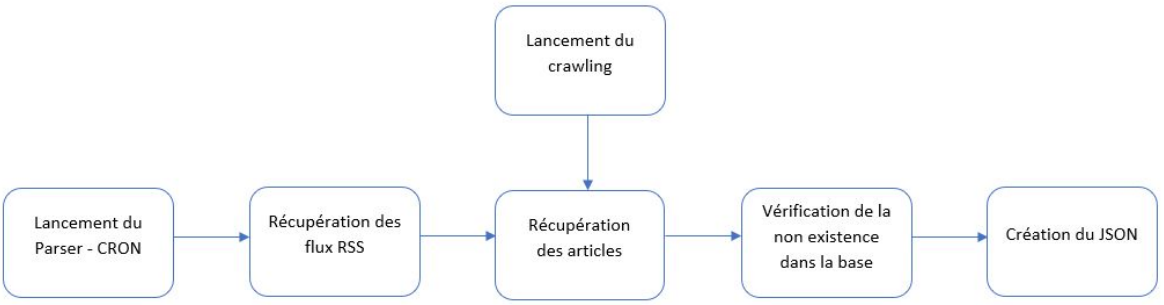


Schéma du workflow

Pour chaque journal, l'un des objectifs est d'obtenir les flux RSS afin de récupérer, chaque jour, les nouveaux articles sortis. L'autre objectif est de crawler le journal afin de récupérer un maximum d'articles déjà présents sur le site et cela afin de créer une base d'articles.

Durant le reste de cette partie, nous prendrons comme exemple le journal "Le Nouvel Observateur". Afin de simplifier le code et le nom des fichiers, sa forme abrégée est : "noob".

Nos algorithmes seront appelés par un fichier parser qui sera exécuté à l'aide d'un CRON une fois par jour. Ils seront donc amenés à traiter plusieurs fois les mêmes articles. Pour détecter les articles déjà traités et les ignorer nous avons utilisé l'algorithme MD5. Le MD5 calcul le hash en se basant sur certaines des métadonnées de l'article (la date de publication, le titre, le nom du journal) et permet d'obtenir une empreinte qui a de grande chance d'être unique.

3.1. Fonctions génériques

Afin de simplifier les codes, nous avons créé des fonctions génériques pouvant être utilisées pour n'importe quel journal que nous avons traité. Ces fonctions sont présentes dans un fichier *"g4_utils_v40.py"*[1].

La fonction "*add_to_index*" permet d'ajouter, à chaque ajout d'un nouvel article, une virgule dans le fichier "*hash_text.csv*" pour y séparer les articles.

[1] Voir Annexe 1

La fonction "*get_hash*" enlève toutes les consonnes présentes dans le titre de l'article et dans le nom du journal.

La fonction "already_exists" vérifie si le nouvel article que l'on veut ajouter n'existe pas déjà à l'aide du fichier "hash_text.csv".

La fonction "create_index" ajoute dans le fichier "*hash_text.csv*" chaque nouvel article. L'article est d'abord transformé par la fonction get_hash avant d'être inséré dans le fichier csv.

La fonction "*create_json*" prenant en paramètre un chemin, une liste d'article, une source (ici "NouvelObs") et l'abréviation du journal (ici "noob"), crée un fichier json pour chaque article.

La fonction "*recovery_article*" crée un dictionnaire contenant les informations nécessaires récupérées pour chaque article.

La fonction "*is_empty*" retourne False si les informations concernant le titre, le contenu et la date de publication d'un article sont présentes. Cela permet de ne récupérer que les articles pouvant ensuite être traités par les autres groupes.

La fonction "*recovery_flux_rss_rss*" permet de parser un url donné afin de pouvoir analyser le contenu d'une page internet.

3.2. Récupération du flux RSS

Le principe d'un flux RSS est de récupérer la liste de tous les nouveaux articles parus récemment. Le plus souvent, un flux RSS est un fichier XML.
Le code pour récupérer les nouveaux articles provenant du flux RSS est dans le ficher "*g4_noob_rss_function.py*"[2]. Ce code est composé de trois fonctions.

[2] Voir Annexe 2

Extrait de la page de flux RSS du Nouvel Observateur

La fonction "*recovery_information_noob*" permet de récupérer toutes les informations nécessaires pour un et un seul article. On utilise la fonction "recovery_flux_url_rss" du fichier "*g4_utils_v40.py*" afin de pouvoir traiter l'article provenant du site internet. On fait appel à la fonction "*recovery_article*" située dans "*g4_utils_v40.py*" afin de transformer les informations récupérées sous forme de dictionnaire. Le résultat de cette transformation est retournée par la fonction.

La page de flux RSS est composée des différentes catégories composant le site. Pour chacune de ces catégories il correspond plusieurs liens contenant ses flux RSS. En effet on peut y accéder via Yahoo ou encore MSN. Dans notre cas, nous voulons juste récupérer le lien XML basique (représenté par le carré orange sur la capture d'écran ci-dessus). Ainsi, la fonction "*recovery_link_new_articles_noob_rss*" permet de récupérer les urls de tous ces différents flux RSS présents sur la page. Ces url sont stockés dans une liste (liste_url) qui est retournée par la fonction.

La fonction "*recovery_new_articles_noob_rss*" permet de récupérer pour chaque url de liste_url la liste de tous les nouveaux articles. Pour chaque article de cette liste, on va récupérer les informations nécessaires grâce à la fonction expliquée ci-dessus ("*recovery_information_noob*"). Ces informations sont ajoutées à une liste (list_json) qui a pour but de contenir toutes les informations pour tous les nouveaux articles. A la fin de cette fonction on fait appel à la fonction "*create_json*" (voir partie 3.1).

On peut retrouver tous les fichiers json ainsi créés en suivant le chemin suivant : data/clean/robot/YYYY-MM-DD/NouvelObs_nouveaux/

3.3. Récupération du crawler

Le but du crawler est de récupérer un maximum d'articles présents sur un site.

Tout comme pour la récupération du flux RSS, le code pour crawler le site est composé de trois fonctions. Le code est situé dans le fichier "*g4_noob_crawler_function.py*"[3].

La fonction "*recovery_information_noob*" est toujours présente afin de récupérer les informations pour un url d'un article donné.

La fonction "*recovery_link_new_articles_noob_crawler*" parcours toutes les différentes catégories du site (ex : politique, culture). Pour chaque catégorie on va récupérer tous les urls des articles des pages 2 à 7 incluses. La fonction renvoie la liste des urls de tous ces articles.

La fonction "*recovery_new_articles_noob_crawler*" récupère la liste de tous les urls des articles en faisant appel à la fonction précédente. Pour chaque url, on fait appel à la fonction "*recovery_information_noob*". La sortie de cette fonction est stockée dans une liste (list_json). A la fin de cette fonction on fait appel à la fonction "*create_json*" (voir partie 3.1).

On peut retrouver tous les fichiers json ainsi créés en suivant le chemin suivant : data/clean/robot/YYYY-MM-DD/NouvelObs_crawler/

3.4. Journaux traités

Légende :

- Existe en local : L
- Découpé en fonction : D
- Testé : T
- Testé sur le serveur : S

[3] Voir Annexe 3

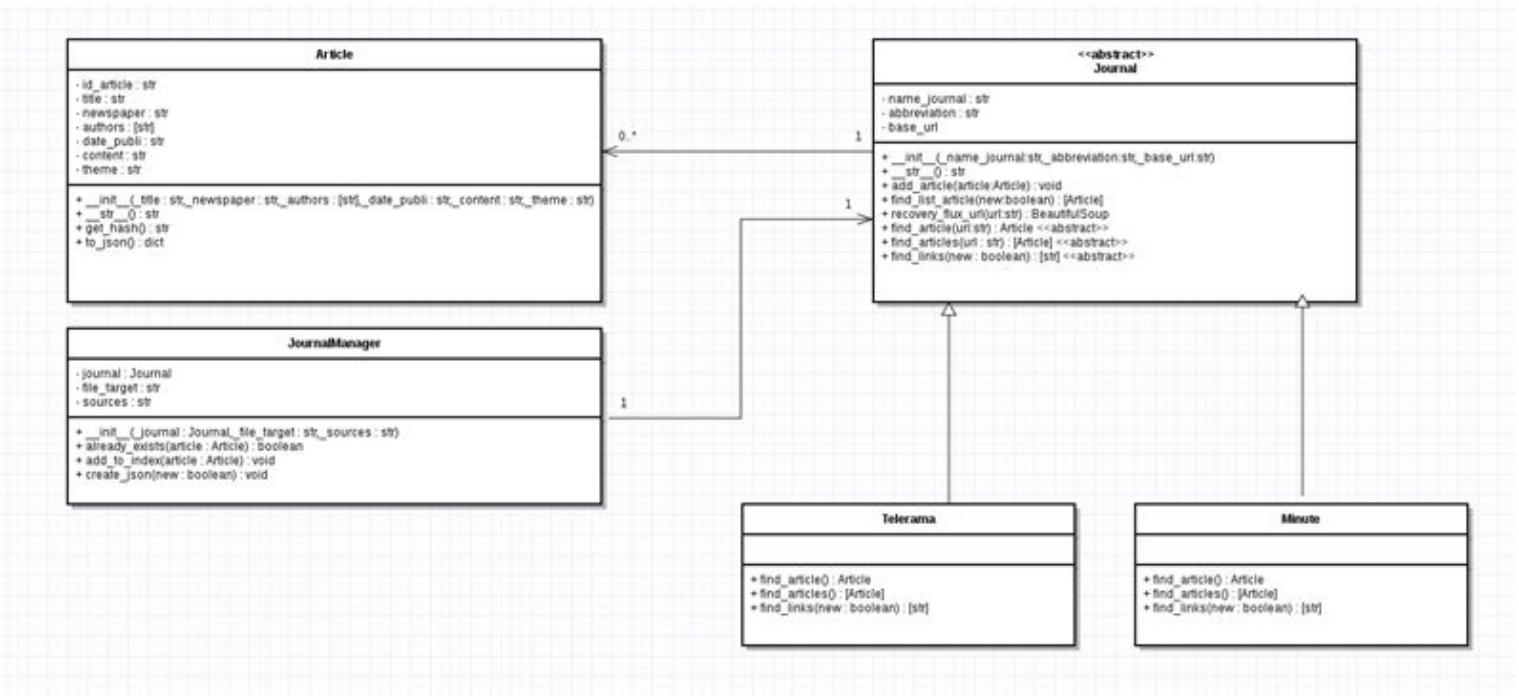
	RSS	Crawler
Futurascience	L D T S	L D T S
Femina	Pas de flux RSS	L D T S
20 minutes	L D T S	L D T S
Gorafi	L D T S	L D T S
La Depeche	L D T S	L D T S
Liberation	L D T S	L D T S
Le Monde	L D T S	L D T S
Le Nouvel Observateur	L D T S	L D T S
Le Figaro	L D T S	L D T S
Le Point	L D T S	L D T S
Telerama	L D T S	L D T S
L'Humanite	L D T S	L D T S
L'Equipe	L D T S	L D T S
La Tribune	L D T S	L D
Science Et Vie	Pas de flux RSS	L D T S

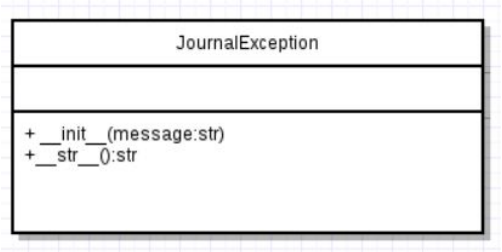
4. Missions complémentaires

Après avoir réalisé l'ensemble des missions principales, nous avons commencé à réaliser les missions complémentaires. Ces missions correspondent à la création de classes, qui permettent de faciliter l'intégration de nouveaux journaux, et à une méthode de connexion automatique.

4.1. Transformation en classes

Diagramme de classes





Pour faciliter l'organisation, la réutilisation et la correction de notre code et que celui-ci soit plus générique, nous avons créé des classes. Dans le cadre de ce projet, nous avons mis en place les classes suivantes : “Article”, “Journal”, “JournalManager” et “JournalException”.

4.1.1. Classe Article

Elle permet d’identifier de manière unique un article. Elle définit deux méthodes : get_hash et to_json qui permettent respectivement de déterminer l'identifiant d'un article et de le représenter sous format json.

4.1.2. Classe Journal

La classe Journal permet de récupérer les nouveaux articles mais aussi les anciens à partir de leur adresse url. Elle est abstraite et est la classe mère de toutes les classes (Equipe, Femina, Figaro, FuturaSciences, Humanite, ...). Elle implante trois méthodes abstraites qui sont :
- la méthode “find_article” : retourne un article à partir de l'adresse url
- la méthode “find_articles” : retourne une liste d'articles à partir de l'adresse url
- la méthode “find_links” : retourne une liste de liens (les liens des anciens ou des nouveaux articles)

Cette classe permet aussi de créer un objet BeautifulSoul avec la fonction “*recovery_flux_url_rss*” présente dans le fichier “*g4_utils_v40.py*” et d'ajouter un objet Article dans un journal.

4.1.3. Classe Journal Manager

La classe Journal Manager permet de créer pour chaque article un fichier json, d'indexer les articles mais aussi de vérifier si un article est déjà récupéré.

4.1.4. Classe Journal Exception

Enfin, nous avons la classe “JournalException” qui peut lever une exception quand on tente de créer une instance de Journal ou quand on ne définit pas l'une des trois méthodes (“find_article”, “find_articles”, “find_links”). Elle peut aussi lever une exception lorsqu’on essaye d'ajouter un objet qui n'est pas un article dans un journal.

4.2. Authentification automatique avec un compte d'abonné

Avant de parcourir les journaux on doit s'identifier pour quelques uns qui sont payants. L'objectif de cette partie est de se connecter automatiquement pour pouvoir accéder aux articles de ces journaux.
Le code de cette partie est composé de quatre fonctions, une fonction pour chaque site. Nous n'avons pas pu écrire une seule fonction car chaque site a ses propres balises pour l'authentification. Dans ce rapport nous allons citer une seule fonction. Le code est situé dans l'annexe 3 : Def connect_ldpch(email,password) .

La fonction "connect_ldpch" prend comme paramètres le login et le mot de passe d'un compte d'abonné pour se connecter. Cette fonction permet de naviguer sur un site web à travers WebDriver en utilisant PhantomJS. Elle permet de récupérer et remplir les champs d'authentification et de valider le formulaire afin de se connecter.

5. Difficultés rencontrées et solutions apportées

Durant ce projet, plusieurs difficultés ont été rencontrées.

Tout d'abord, nous avons eu des problèmes dus à une connexion au wifi instable. En effet, dans les salles où nous nous trouvions nous n'avions pas ou peu accès aux réseaux spécifiques à l'université (eduroam et ups). Par conséquent, afin de pallier au problème nous avons utilisé la connexion partagée de nos smartphones.

Un autre problème est que certains sites tel que Femina ne possèdent pas de flux RSS disponibles. Ainsi la solution utilisée a été de réaliser uniquement le crawler. Celui-ci sera lancé tous les jours en ne gardant que la première page ou les deux premières pages. Grâce à la table de hachage, les articles déjà présents dans la base ne seront pas récupérés une seconde fois. Ainsi on pourra récupérer les nouveaux articles.

Par ailleurs, nous avons eu des difficultés au niveau de la récupération d'articles. En effet, tous les articles d'un même site ne possèdent pas obligatoirement les mêmes balises ou attributs. Ainsi, pour un journal donné, si pour certains articles il existe un attribut "datetime" dans lequel on retrouve la date, dans d'autres articles cet attribut n'est pas présent. Par conséquent, il faut pour chaque journal trouver la balise qui existe contenant l'information que l'on souhaite obtenir.

Nous avons également eu quelques conflits avec GitHub d'origine inconnue. Par exemple, GitHub Desktop indiquait que nous avions transformé des fichiers alors que ce n'était pas le cas. Afin de résoudre ce problème nous avons transféré les bons fichiers via une clé usb puis une personne réglait le problème en revenant à une arborescence propre.

Nous avons également été limités à cause de certains journaux payants. En effet, sur certains sites, nous n'avons droit qu'à un certain nombre de visite. De ce fait, nous étions par moment bloqués pour lancer notre code. Afin de résoudre ce problème, soit un autre membre du groupe lançait le code soit nous attendions le lendemain pour le relancer et travaillions sur autre chose le reste de la journée.

Nous avons également rencontré des problèmes d'encodage avec quelques articles de L'Equipe, qui étaient encodé différemment des autres articles. Nous avons réussi à contrer ce problème en indiquant que si il y a un problème d'encodage pour ces articles, nous ne les récupérons pas. Il s'agit de très peu d'article, moins de dix articles, parmis les centaines récupérés.

6. Conclusion

Les objectifs des premières missions ont quasiment tous été réalisés. Tous les journaux sont traités au jour le jour (ce qui correspond à peu près 26 Mo), et il ne manque que les crawler qui sont en phase de test. De plus, certaines fonctions de la deuxième version ont été réalisées mais pas implémentées. Nous avons essayé tout au long du projet de prendre en compte toutes les demandes des autres groupes afin de leurs donner des données aussi pertinentes que possible.

Si nous avions eu plus de temps, nous aurions implémenté sur les autres journaux la programmation orientée objet, et le fait de se connecter automatiquement. De plus, les tests étant assez longs, nous aurions pu aller plus profondément dans les tests afin de mieux vérifier l'efficacité de nos algorithmes.

Ce projet nous a permis de mettre en application nos compétences en gestion de projet vues tout au long de notre cursus, en mettant en place une méthode AGILE. De plus, cela nous a permi d'améliorer nos compétences en python, notamment la maîtrise de l'extraction d'informations d'une page internet, et l'interaction avec cette dernière.

Annexe 1 : Fichier g4_utils

```
# -*- coding: utf-8 -*-
"""
Groupe 4
Céline MOTHES
Morgan SEGUELA
V1 : function create_json
V2 : add function add_to_index, get_hash, already_exists, create_index
V3 : add function recovery_flux_urss, recovery_article
V31 : change a variable
V32 : modification of the recovery_article function
V33 : modification hash + function create_json
V34 : deleting date formatting in the create json function
V40 : Add the function is_empty
"""
import csv
import datetime as date
import json
import os
import bs4
import requests
from hashlib import md5
import unidecode
import re


def add_to_index(date_publi, text, newspaper):
  hash_text = get_hash(date_publi, text, newspaper)
  with open("hash_text.csv", "a") as f:
    f.write(hash_text + ",")


def get_hash(date_publi, text, newspaper):
  """create a hash from the date, the title, and the newspaper to find
  if an article already exists
  Arguments:
    date {string} -- date of the article
    text {string} -- title of the article
    newspaper {string} -- name of the newspaper

  Returns:
    string -- a hash of the article
  """

  hash = md5("{} {} {}".format(date_publi, text, newspaper).encode())
  return hash.hexdigest()
```

```
def already_exists(date_publi, text, newspaper):
  """create a test to see if the article entered already exists
  Arguments:
    date {string} -- date of the article
    text {string} -- title of the article
    newspaper {string} -- name of the newspaper

  Returns:
    boolean -- False: Doesn't exist | True: Does exist
  """

  hash_text = get_hash(date_publi, text, newspaper)
  with open("hash_text.csv", "r") as f:
    csv_reader = csv.reader(f, delimiter=",")
    already_existing_hash = csv_reader.__next__()[:-1]
  return hash_text in already_existing_hash


def create_index():
  """Create the index for all the article saved
  """
  try:
    source = "/var/www/html/projet2018/data/clean/robot/"
    dates_extract = os.listdir(source)

    hash_text = []
    for date_extract in dates_extract:
      source_date = source + date_extract + "/"
      for newspaper in os.listdir(source_date):
        source_newpaper = source_date + newspaper + "/"

        for article in os.listdir(source_newpaper):
          u8 = "utf-8"
          with open(source_newpaper + article, "r", encoding=u8) \
              as f:
            data = json.load(f)
            title = data["title"]
            date_publi = data["date_publi"]
            newspaper = data["newspaper"]
          hash_text.append(get_hash(date_publi, title, newspaper))
    hash_text = list(set(hash_text))

    with open("hash_text.csv", "a") as f:
      f.write(",".join(hash_text)+",")
    print("creer")

  except:
    with open("hash_text.csv", "w") as f:
      f.write(",")
      print("creer")
```

```
def create_json(file_target, list_article, sources, abbreviation):
  """
  Entree:
     file_target: string containing the path of the folder
     sources: string containing folder name
     list_article: list containing new articles
     abbreviation:string containing the journal abbreviation
  Exit:
     one json file per item

  For each article, the function creates a json file named after it:
     art_abreviation_numeroArticle_datejour_robot. json.
  It places the json file in the folder corresponding to the journal
  if it exists otherwise it creates it.
  """
  os.makedirs(file_target+sources, exist_ok=True)
  list_file = os.listdir(file_target+sources)
  num_max = 0
  if list_file:
     list_num = []
     for file in list_file:
        delimiter = file.split("_")
        list_num.append(delimiter[2])
     for num in list_num:
        num_max = max(num_max, int(num))
  ii = num_max + 1

  cur_date = date.datetime.now().date()
  for article in list_article:
     if not already_exists(article["date_publi"], article["title"],
                           article["newspaper"]):
        add_to_index(article["date_publi"], article["title"],
                     article["newspaper"])
        if "/" in sources:
           file_art = file_target + sources + "art_" + abbreviation + "_"\
              + str(ii) + "_" + str(cur_date) + "_robot.json"
        else:
           file_art = file_target + sources + "/" + "art_" + abbreviation\
            + "_" + str(ii) + "_" + str(cur_date) + "_robot.json"
        with open(file_art, "w", encoding="UTF-8") as fic:
           json.dump(article, fic, ensure_ascii=False)

        ii += 1
```

```
def recovery_article(title, newspaper, authors, date_publi, content, theme):
```

```
    """
    Arguments:
        title : string
        newspaper : string
        authors : list
        date_publi : string
        content : string
        theme : string
    Return : dictionary containing title, newspaper,
    """
    for ii in range(len(authors)):
        authors[ii] = unidecode.unidecode(authors[ii])

    content = re.sub(r"\s\s+", " ", content)
    content = re.sub(r"\s", " ", content)

    new_article = {
            "id_art": get_hash(date_publi, title, newspaper),
            "title": unidecode.unidecode(title),
            "newspaper": unidecode.unidecode(newspaper),
            "author": authors,
            "date_publi": date_publi,
            "content": unidecode.unidecode(content),
            "theme": unidecode.unidecode(theme)
        }
    return(new_article)



def is_empty(article):
    """
    Argument :
        article : dictionary
    Add the article if the content, the publication date and the title
        are not empty
    """
    if (article["content"] != ""
            and article["title"] != ""
            and article["date_publi"] != ""):
        return False

    return True




def recovery_flux_url_rss(url_rss):
    """
    Arguments:
```

```
    string containing the url of the rss feed
  Return :
    object containing the parse page
  The function parse the page with beautifulsoup
  """
  req = requests.get(url_rss)
  data = req.text
  soup = bs4.BeautifulSoup(data, "lxml")
  return(soup)


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

Annexe 2 : Fichier g4_noob_rss_function.py

```
# Groupe 4
# DELOEUVRE Noémie

import g4_utils_v40 as utils
import re
from datetime import datetime
import datetime as date


def recovery_information_noob(url_article):
    """
        Arguments:
            - url of one article
        Returns:
            - informations of the article
    """
    soup_article = utils.recovery_flux_url_rss(url_article)

    title = soup_article.title.get_text()

    # Retrieval of publication date
    find_date = soup_article.find('time', attrs={"class": "date"})
    for a in find_date.find_all('a'):
        find_valeur = re.compile('[0-9]{4}\/[0-9]{2}\/[0-9]{2}')
        for valeur in find_valeur.finditer(str(a.get("href"))):
            date_p = valeur.group(0)
            date_p = datetime.strptime(date_p, "%Y/%m/%d")\
                .strftime("%Y-%m-%d")

    # Retrieval of the author of the article
    author = []
    for div in soup_article.find_all('div'):
        for valeur in re.finditer('author', str(div.get("class"))):
            author.append(div.p.span.get_text())

    # Retrieval of the artical theme
    theme = ""
    for nav in soup_article.find_all('nav'):
        if nav.get("class") == ['breadcrumb']:
            for ol in nav.find_all('ol'):
                for a in ol.find_all('a'):
                    theme = a.get_text()

    # Retrieving the content of the article
    contents = ""
    for div in soup_article.find_all('div'):
```

```
        for valeur in re.finditer('body', str(div.get("id"))):
            for aside in div.find_all('aside'):
                for p in aside.find_all('p'):
                    p.string = ""
            for p in div.find_all('p'):
                for a in p.find_all('a'):
                    if a.get("class") == ['lire']:
                        a.string = ""
                for img in p.find_all('img'):
                    p.string = ""
                contents += p.get_text() + " "

    article = utils.recovery_article(title, 'NouvelObservateur',
                                     author, date_p, contents, theme)
    return(article)


def recovery_link_new_articles_noob_rss(url_rss):
    """
        Arguments:
            - url of the page containing feed links for
            the different categories
        Returns :
            - list of urls of the different categories

    """
    soup = utils.recovery_flux_url_rss(url_rss)

    liste_url = []
    # Retrieving all urls of new RSS feeds of different categories
    for a in soup.find_all('a'):
        if a.get("class") == ['sprite-rss', 'sp-rss']:
            for valeur in re.finditer('www', str(a.get("href"))):
                liste_url.append(a.get("href"))

    return(liste_url)


def recovery_new_articles_noob_rss(file_target="data/clean/robot/" +
                                   str(date.datetime.now().date()) + "/"):
    """
        Returns:
            - creation of a json for each new article
    """
    file_json = []
    # Each url is analized one by one
    list_url = recovery_link_new_articles_noob_rss("http://www.nouvelobs." +
                                                   "com/rss/")
    for url in list_url:
        soup_url = utils.recovery_flux_url_rss(url)
        items = soup_url.find_all("item")
        article_noob = []
```

```
        # We're picking up every new article in a list
        for item in items:
            link_article = re.search(r"<link/>(.*)", str(item))[1]
            link_article = link_article.split("<description>")
            link_article = link_article[0]
            article_noob.append(link_article)
            for valeur in re.finditer("\/galeries\-photos\/", link_article):
                article_noob.remove(link_article)
        # Each article is analized one by one
        for article in article_noob:
            new_article = recovery_information_noob(article)
            if utils.is_empty(new_article) is False:
                file_json.append(new_article)

    utils.create_json(file_target, file_json, "NouvelObs_nouveaux/",
                      "noob")


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

Annexe 3 : Fichier g4_noob_crawler_function.py

```
# Groupe 4
# DELOEUVRE Noémie

import g4_utils_v40 as utils
import re
from datetime import datetime
import datetime as date


def recovery_information_noob(url_article):
    """
        Arguments:
            - url of one article
        Returns:
            - informations of the article
    """
    soup_article = utils.recovery_flux_url_rss(url_article)

    title = soup_article.title.get_text()

    # Retrieval of publication date
    find_date = soup_article.find('time', attrs={"class": "date"})
    for a in find_date.find_all('a'):
        find_valeur = re.compile('[0-9]{4}\/[0-9]{2}\/[0-9]{2}')
        for valeur in find_valeur.finditer(str(a.get("href"))):
            date_p = valeur.group(0)
            date_p = datetime.strptime(date_p, "%Y/%m/%d")\
                .strftime("%Y-%m-%d")

    # Retrieval of the author of the article
    author = []
    for div in soup_article.find_all('div'):
        for valeur in re.finditer('author', str(div.get("class"))):
            author.append(div.p.span.get_text())

    # Retrieval of the artical theme
    theme = ""
    for nav in soup_article.find_all('nav'):
        if nav.get("class") == ['breadcrumb']:
            for ol in nav.find_all('ol'):
                for a in ol.find_all('a'):
                    theme = a.get_text()

    # Retrieving the content of the article
    contents = ""
    for div in soup_article.find_all('div'):
```

```
        for valeur in re.finditer('body', str(div.get("id"))):
            for aside in div.find_all('aside'):
                for p in aside.find_all('p'):
                    p.string = ""
            for p in div.find_all('p'):
                for a in p.find_all('a'):
                    if a.get("class") == ['lire']:
                        a.string = ""
                for img in p.find_all('img'):
                    p.string = ""
                contents += p.get_text() + " "

    article = utils.recovery_article(title, 'NouvelObservateur',
                                     author, date_p, contents, theme)
    return(article)


def recovery_link_new_articles_noob_crawler():
    """
        Arguments:
            - url of the page containing feed links for
            the different categories
        Returns :
            - list of urls of the different categories
    """
    list_category = ["politique", "monde", "economie", "culture",
                     "editos-et-chroniques", "debat"]

    article_noob = []
    for cat in list_category:
        # We retrieve the URL feeds for each page of article
        # Each HTML-coded article is analyzed with beautiful soup
        for i in range(2, 8):
            url_rss_noob = "http://www.nouvelobs.com/" + cat +\
                "/page-" + str(i) + ".html"

            soup_url = utils.recovery_flux_url_rss(url_rss_noob)

            # We retrieve all the articles for a given page
            for h3 in soup_url.find_all('h3'):
                if h3.get("class") == ['title']:
                    for valeur in re.finditer('^\/', str(h3.a.get("href"))):
                        new_article = "http://www.nouvelobs.com" +\
                            h3.a.get("href")
                        article_noob.append(new_article)

    return(article_noob)


def recovery_new_articles_noob_crawler(file_target="data/clean/robot/" +
                                       str(date.datetime.now().date()) + "/"):
    """
```

```
        Returns:
            - creation of a json for each new article
    """

    file_json = []
    article_noob = recovery_link_new_articles_noob_crawler()

    # Each article is analized one by one
    for article in article_noob:
        new_article = recovery_information_noob(article)
        if utils.is_empty(new_article) is False:
            file_json.append(new_article)

    utils.create_json(file_target, file_json, "NouvelObs_crawler/",
                      "noob")


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

Annexe 4 : Fonction def connnect_ldpch

```
def connect_ldpch(email, passwrd):
    url = "https://www.ladepeche.fr/"
    driver = webdriver.PhantomJS(executable_path="/usr/local/bin/phantomjs")
    driver.get(url)
    driver.set_window_size(1120, 550)
    email = "watchnewsapp@gmail.com"
    passwrd = "projetsid2018"
    # Cliquer sur connexion
    driver.find_element_by_xpath(('//a[contains(@onclick,"login")]')).click()
    time.sleep(3)
    # Locliser et remplir les champs
    driver.find_element_by_id("user_login").send_keys(email)
    driver.find_element_by_id("user_pass").send_keys(passwrd)
    time.sleep(3)
    # Click sur connexion
    driver.find_element_by_id('submit-login').click()
    time.sleep(10)
    # Saves the screenshot
    driver.save_screenshot("/Users/nabil/Desktop/Projet/testLogin/ladepecheee.png")
    driver.quit()
```