Jour 3 : Introduction au scraping web avec *requests* et *BeautifulSoup*

Sommaire

Introduction : ***3***

Partie 0 : Prérequis ***3***

Partie 1 : Introduction à requests pour faire des requêtes HTTP ***4***

Exercice 1 : Effectuer une requête GET **4**

Exercice 2 : Paramètres de requête (Query parameters) **4**

Exercice 3 : Gestion des codes de statut **5**

Exercice 4 : Introduction aux headers **5**

Partie 2 : Introduction à BeautifulSoup pour faire du scraping web ***6***

Exercice 0 : Création d'un objet *BeautifulSoup* **6**

Exercice 1 : Trouver une balise unique **7**

Exercice 2 : Naviguer et extraire des balises HTML spécifique **7**

Exercice 3 : Extraire des attributs de balises HTML **8**

Exercice 4 : Naviguer à l'aide des sélecteurs CSS **8**

Exercice 5 : Extraire le contenu de différentes balises HTML **9**

Exercice 6 : Extraire le contenu d'un tableau HTML **10**

Partie 3 : Scraping avancé ***11***

Exercice 1 : Navigation avancée **11**

Exercice 2 : Récupération d'informations en deux temps **12**

Exercice 3 : Navigation récursive **12**

Partie 4 : Scraping en conditions réelles ***13***

Exercice 1 : Analyser un contenu spécifique **13**

Exercice 2 : Récupération de contenu imbriqué **14**

Exercice 3 : Récupérer une liste de contenu spécifique **14**

Exercice 4 : Gestion de la pagination **15**

Exercice 5 : Extraction de données par catégorie **16**

Exercice 6 : Stockage des informations et agrégation de masse **17**

Exercice 7 : Utilisation des données récupérées.....17

Going Further : Mini-projet 19

Introduction :

Dans le domaine de la Big Data et du développement web, la capacité de rassembler et de traiter des informations depuis internet est inestimable. Le scraping web, une méthode d'automatisation de l'extraction de données à partir de sites web, est une compétence cruciale pour tout professionnel de la Data. Elle nous permet de convertir des données web non structurées en une forme structurée qui peut être analysée et utilisée dans diverses applications.

Partie 0 : Prérequis

Pour cette journée, vous aurez besoin des librairies Python suivantes :

- pandas, que vous avez déjà utilisé lors de la journée précédente,
- requests, une librairie HTTP simple et intuitive pour Python,
- Beautiful Soup, une librairie pour extraire des données de fichiers HTML et XML avec Python.

Comme vu hier, ajoutez ces librairies au fichier *requirements.txt* et installez-les dans un nouvel environnement virtuel dédié à la journée .

Partie 1 : Introduction à *requests* pour faire des requêtes HTTP

Dans le monde du scraping web et de la collecte de données, `requests` est une bibliothèque incontournable pour effectuer des requêtes HTTP en Python. Simple à utiliser mais puissante, elle permet de récupérer facilement le contenu web grâce à son API intuitive. Cette section vous introduit aux bases de la réalisation de requêtes GET et de la gestion des réponses. En maîtrisant cette bibliothèque, vous aurez les bases pour réaliser des projets de scraping web sophistiqués. Toutes les informations dont vous aurez besoin pour réaliser les exercices suivants se trouvent dans la documentation officielle de requests.

Exercice 1 : Effectuer une requête GET

Prototype : *get_request*(url: str) -> (int, str)
Rendu : partie1.py

Écrire une fonction *get_request* qui prend en paramètre une URL et utilise `requests` pour effectuer une requête GET sur cette URL. Retourner dans un tuple le code de statut de la réponse ainsi que le contenu en format JSON.

Vous pouvez utiliser l'API de votre choix pour tester votre fonction, comme REST Countries.

Exercice 2 : Paramètres de requête (Query parameters)

Prototype : *get_countries_info*(country_codes: list, info: list) -> (int, str)
Rendu : partie1.py

Écrire une fonction *get_countries_info* prenant en paramètre une liste de code pays et une liste d'informations à récupérer. En utilisant l'API REST Countries, trouver l'URL qui permet de filtrer les pays via les codes pays et filtrer ceux-ci avec la liste prise en paramètre. Trouver le paramètre de requête qui permet de filtrer les informations (capitale, langues, …) retournées par l'API. Retourner dans un tuple le code de statut de la réponse ainsi que le contenu en format JSON.

Exercice 3 : Gestion des codes de statut

Prototype : *handle_request_status*(url: str) -> int | str
Rendu : partie1.py

Écrire une fonction *handle_request_status* qui prend une URL. Faire une requête POST sur l'URL Vérifier le code de statut de la réponse, si la requête est un succès, retournez le code de statut, sinon trouver la fonction de `requests` qui permet de lever une exception et retourner celle-ci sous forme de string à l'aide d'un bloc `try-except`.

Indice : Vous pouvez tester votre fonction grâce au site HTTPBin et particulièrement à l'URL https://httpbin.org/status en passant une liste de statut séparés par une virgule dans l'URL.

Exercice 4 : Introduction aux headers

Prototypes :

- *send_query_parameters*(params: dict) -> dict
- *send_headers(headers: dict) -> str*

Rendu : partie1.py

Écrire une fonction *send_query_parameters* qui prend un dictionnaire de paramètres de requête (query parameters). Faire une requête GET sur l'URL https://httpbin.org/response-headers en passant le dictionnaire en paramètres de requête. Retourner les headers de la réponse sous forme de `dict`.

Écrire une fonction *sent_headers* qui prend un dictionnaire de headers. Faire une requête GET sur l'URL https://httpbin.org/headers en y ajoutant les headers pris en paramètre. Retourner le contenu dans la clé `headers` de la réponse.

Indice : Il arrive parfois que le header « User-Agent » de base de requests soit banni sur certains site. Il est possible de le changer à une autre valeur de votre choix représentant de préférence l'identité de votre programme (« datapool_day2 » par exemple).

Partie 2 : Introduction à *BeautifulSoup* pour faire du scraping web

BeautifulSoup est une bibliothèque Python conçue pour analyser les documents HTML et XML, la rendant indispensable pour le scraping web. Elle transforme les codes sources de pages web en un arbre navigable d'objets, permettant une extraction simplifiée des données. Les deux parties suivantes vous guideront à travers les essentiels de BeautifulSoup pour extraire de structures simples ou complexes, les informations dont vous avez besoin. En apprenant à utiliser BeautifulSoup, vous gagnerez la capacité de transformer des pages HTML chaotiques en un ensemble de données structuré prêt pour l'analyse ou le stockage. Toutes les informations dont vous aurez besoin pour réaliser les exercices suivants se trouvent dans la documentation officielle de BeautifulSoup.

Pour cette partie, utilisez le fichier « example.html » fourni avec le sujet pour tester votre code.

Exercice 0 : Création d'un objet *BeautifulSoup*

Prototype : *create_bs_obj*(file: str) -> BeautifulSoup
Rendu : partie2.py

Écrire une fonction *create_bs_obj* prenant en paramètre un nom de fichier HTML. Ouvrir ce fichier et créer un objet BeautifulSoup à partir de celui-ci. Retourner l'objet créé.

Indice : Vous pouvez réutiliser cette fonction pour tous les exercices suivants plutôt que d'avoir à copier-coller le code. C'est un principe appelé communément DRY (Don't Repeat Yourself).

Exercice 1 : Trouver une balise unique

Prototype : *find_title*(file: str) -> str
Rendu : partie2.py

Écrire une fonction *create_bs_obj* prenant en paramètre un nom de fichier HTML. Charger ce fichier, trouver et retourner le titre (balise `title`) de la page. Vous devez convertir le retour en string.

Exemple :

```
file = 'resources/example.html'
title = find_title(file)
print(title)
```

```
(.venv) →  Jour3 git:(main) ✗ python3 partie2.py | cat -e
<title>BeautifulSoup Practice Page</title>$
(.venv) →  Jour3 git:(main) ✗
```

Exercice 2 : Naviguer et extraire des balises HTML spécifique

Prototype : *find_paragraphs*(file: str) -> list[str]
Rendu : partie2.py

Écrire une fonction *find_paragraphs* prenant en paramètre un nom de fichier HTML. Charger ce fichier, trouver et récupérer toutes les balises de paragraphe `<p>`. Convertir toutes les balises en string et les retourner dans une liste.

Exemple :

```
file = 'resources/example.html'
paragraphs = find_paragraphs(file)
for paragraph in paragraphs:
    print(paragraph)
```

```
(.venv) →  Jour3 git:(main) ✗ python3 partie2.py | cat -e
<p class="info">This page is designed to help you practice web scraping with BeautifulSoup.</p>$
<p>Hello, BeautifulSoup! Let's navigate the HTML tree.</p>$
<p>Extracting all links from the page is fun!</p>$
<p>Using CSS selectors makes selecting elements straightforward.</p>$
<p>Extracting clean text is essential for data analysis.</p>$
(.venv) →  Jour3 git:(main) ✗
```

Exercice 3 : Extraire des attributs de balises HTML

Prototype : *find_links*(file: str) -> list[str]
Rendu : partie2.py

Écrire une fonction *find_links* prenant en paramètre un nom de fichier HTML. Charger ce fichier, trouver et retourner dans une liste tous les liens contenus dans les balises `<a>`.

Indice : Les liens dans les balises `<a>` sont toujours contenus dans l'attribut `href`.

Exemple :

```
file = 'resources/example.html'
links = find_links(file)
print(links)
```

```
(.venv) → Jour3 git:(main) × python3 partie2.py | cat -e
['https://example.com/page1', 'https://example.com/page2', 'https://example.com/page3']$
(.venv) → Jour3 git:(main) ×
```

Exercice 4 : Naviguer à l'aide des sélecteurs CSS

Prototype : *find_elements_with_css_class*(file: str, class_name: str) -> list[str]
Rendu : partie2.py

Écrire une fonction *find_elements_with_css_class* prenant en paramètre un nom de fichier HTML et un nom de classe CSS. Charger le fichier, trouver et retourner dans une liste tous les éléments/balises possédant la classe CSS prise en paramètre. Convertir toutes les balises en string et les retourner dans une liste.

Indice : Dans le fichier `example.html`, il y a une classe `info` qui est associée à plusieurs éléments différents. Il est possible de tester votre exercice grâce à celle-ci.

Exemple

```
file = 'resources/example.html'
css_classes = find_elements_with_css_class(file, 'info')
for class_elem in css_classes:
    print(class_elem)
```

```
(.venv) → Jour3 git:(main) × python3 partie2.py | cat -e
<p class="info">This page is designed to help you practice web scraping with BeautifulSoup.</p>$
<div class="info">Content with class "info" to demonstrate CSS selector usage.</div>$
(.venv) → Jour3 git:(main) ×
```

Exercice 5 : Extraire le contenu de différentes balises HTML

Prototype : *find_headers*(file: str) -> list[str]
Rendu : partie2.py

Écrire une fonction *find_headers* prenant en paramètre un nom de fichier HTML. Charger ce fichier, extraire et retourner une liste contenant tous les textes contenu dans les éléments d'en-tête (`<h1>`, `<h2>`, etc.).

Indice : Vous pouvez utiliser re.compile et créer une regex permettant de matcher avec tous les headers.

Exemple :

```
file = 'resources/example.html'
headers = find_headers(file)
print(headers)
```

```
(.venv) →  Jour3 git:(main) × python3 partie2.py | cat -e
['Introduction to BeautifulSoup', 'Learning Objectives', 'Working with CSS Selectors', 'Extracting Text']$
(.venv) →  Jour3 git:(main) ×
```

Exercice 6 : Extraire le contenu d'un tableau HTML

Prototype : *extract_table*(file: str) -> list[dict]
Rendu : partie2.py

Écrire une fonction *extract_table* qui prenant en paramètre un nom de fichier HTML. Dans le fichier HTML se trouvera un seul et unique tableau HTML contenant des informations sur des fruits. Ce tableau possèdera 3 colonnes : ` name`, `color` et `price`.
Charger le fichier, analyser le tableau HTML et extraire les informations dans une liste de dictionnaires, où chaque dictionnaire représente un fruit. Faire en sorte que le prix soit de type `float` dans la structure de retour.

Exemple :

```python
file = 'resources/example.html'
fruits = extract_table(file)
for fruit in fruits:
    print(fruit)
```

```
(.venv) →  Jour3 git:(main) × python3 partie2.py | cat -e
{'name': 'Apple', 'color': 'Red', 'price': 1.99}$
{'name': 'Banana', 'color': 'Yellow', 'price': 0.99}$
{'name': 'Cherry', 'color': 'Red', 'price': 2.99}$
(.venv) →  Jour3 git:(main) ×
```

Partie 3 : Scraping avancé

Note importante sur la gestion des requêtes :

À mesure que nous approfondissons les techniques de scraping web plus avancées, il est crucial d'utiliser nos ressources de manière responsable. Envoyer trop de requêtes à un site web dans un court laps de temps peut surcharger le serveur du site et pourrait conduire au blocage de votre adresse IP.

Pour atténuer ce risque et réduire le nombre de requêtes effectuées lors des tests, un script utilitaire appelé « html_utils.py », contant une fonction « fetch_html », est fourni avec le sujet. La fonction permet de **mettre en cache** les résultats de vos requêtes HTTP. Lorsque vous demandez à nouveau la même URL, elle vérifie d'abord si elle a déjà été récupérée et enregistrée. Si c'est le cas, elle récupère le contenu HTML à partir d'un fichier local au lieu de faire une autre requête au site web.

Cela permet une **réduction de la charge** en réduisant considérablement la charge sur le serveur du site ciblé et diminue la probabilité que votre IP soit bloquée en raison de trop nombreuses requêtes.

Pour tous les exercices à venir, sauf indication contraire, veuillez utiliser la fonction « fetch_html » pour récupérer les pages web au lieu d'utiliser directement la bibliothèque « requests ». Cette pratique garantira que vous puissiez continuer à tester et affiner vos scripts de scraping sans interruption et dans le respect des bonnes pratiques du scraping web.

Attention : Vous ne devez pas inclure le fichier utilitaire dans votre rendu !

Exercice 1 : Navigation avancée

Prototype : *find_links_in_paragraphs*(url: str) -> list[str]
Rendu : partie3.py

Écrire une fonction *find_links_in_paragraphs* qui prend une URL en paramètre. Appeler cette URL avec `requests` et récupérer le contenu de la réponse si la requête est un succès, sinon lever une erreur.
Analyser le contenu de la page pour récupérer tous les liens contenu dans un paragraphe. Retourner ceux-ci dans une liste.

Exemple d'URL : la page wikipedia des caméléons

Exercice 2 : Récupération d'informations en deux temps

Prototype : *download_images*(url: str, folder: str, max: int | None = None)
Rendu : partie3.py

Écrire une fonction *download_images* qui prend une URL Wikipedia en paramètre et un chemin vers un dossier. Appeler cette URL avec `fetch_html`, trouver et télécharger toutes les images, n'étant pas des images statiques, dans le dossier pris en paramètre.
La fonction prend un paramètre optionnel « max », s'il vaut « None » (valeur par défaut), télécharger toutes les images, sinon arrêtez-vous à « max » images.

Exemple d'URL : la page wikipedia des caméléons

Indice : Les images statiques sont celles dont la source commence par `/static/`. **Attention**, les URLs d'images ne sont pas complètes, il faut que vous trouviez quel pattern ajouter devant chaque URL. Il est facile d'enregistrer une image obtenue via une requête HTTP en écrivant directement le binaire de celle-ci dans un fichier.

Important : L'user-agent par défaut de `requests` est très limité sur le site Wikipedia, modifiez le header pour mettre votre propre user-agent afin d'avoir accès à toutes les ressources Wikipedia. Pensez également à respecter le robots.txt et à faire des petites pauses (avec sleep par exemple) entre chaque requête pour ne pas surcharger le site.

Exercice 3 : Navigation récursive

Prototype : *recursive_navigation*(url: str, nb: int) -> list[str]
Rendu : partie3.py

Écrire une fonction *recursive_navigation* qui prend une URL du Wikipedia Français et un nombre. Appeler cette URL avec `fetch_html` et récupérer le nb^{ième} lien contenu dans un paragraphe et qui réfère à une autre page du Wikipedia Français. Appeler ce lien et répéter l'opération en enlevant 1 à `nb` à chaque itération et ce jusqu'à ce que `nb` vaille 0.
Retourner une liste contenant tous les liens appelés dans l'ordre d'appel.

Indice : Un lien faisant référence à une autre page du Wikipedia Français est un lien qui commence par `/wiki`.

Partie 4 : Scraping en conditions réelles

Dans cette partie, nous allons utiliser le site Books to Scrape, une librairie simulée conçue pour l'entrainement au scraping. Ce site présente un environnement sûr et légal pour perfectionner vos compétences en scraping. Il présente une large gamme de livres, chacun avec des détails tels que le titre, le prix, la notation et la disponibilité, répartis sur plusieurs pages et catégories. Cette complexité en fait un candidat idéal pour pratiquer la récupération de données, en particulier le traitement de la pagination et l'extraction de données d'une structure HTML complexe.

Note sur la spécificité du scraping web :

Le scraping web consiste à extraire des données à partir de sites web, et il est important de comprendre que chaque site possède une structure HTML unique. En conséquence, le code que vous écrivez pour extraire des données d'un site web est souvent spécialement adapté à l'architecture de ce site.
Si votre projet implique le scraping de plusieurs sites web, vous devrez probablement développer un script de scraping séparé pour chaque site. Cela s'explique par le fait que les sites web peuvent avoir des architectures HTML très différentes, et peuvent même charger des données de manière dynamique avec JavaScript.

Exercice 1 : Analyser un contenu spécifique

Prototype : *get_one_book*() -> dict
Rendu : partie4.py

Dans un navigateur web, aller sur la page d'accueil de Books to Scrape et inspecter le code HTML du premier livre affiché afin de trouver dans quels éléments HTML se trouvent les informations suivantes : le titre(string), la note (nombre entier, int), et le prix (nombre décimal, float).

Suite à cette analyse, écrire une fonction *get_one_book* qui récupère les informations du premier livre. Retourner celles-ci dans un dictionnaire avec les clés suivantes : `title`, `rating` et `price`.

Exemple :

```
book = get_one_book()
print(book)
```

```
(.venv) →  Jour3 git:(main) ✗ python3 partie4.py | cat -e
Loading HTML from the cache$
{'title': 'A Light in the Attic', 'rating': 3, 'price': 51.77}$
(.venv) →  Jour3 git:(main) ✗
```

Exercice 2 : Récupération de contenu imbriqué

Prototype : *get_one_book_complete*() -> dict
Rendu : partie4.py

En continuant l'analyse de la structure HTML du premier livre, trouver le lien vers la page détaillée du livre et analyser cette nouvelle page pour trouver l'élément HTML contenant la description du livre.

Écrire une fonction *get_one_book_complete* qui récupère les mêmes informations que dans l'exercice précédent mais également la description du livre. Retourner les informations dans un dictionnaire avec les clés suivantes : `title`, `rating`, `price` et `description`.

Exemple :

```
book_complete = get_one_book_complete()
print(book_complete)
```

```
(.venv) → Jour3 git:(main) × python3 partie4.py | cat -e
Loading HTML from the cache$
Loading HTML from the cache$
{'title': 'A Light in the Attic', 'rating': 3, 'price': 51.77, 'description': "It's hard to imagine a world without A Light in the Attic. This now-classic collection of poetry
and drawings from Shel Silverstein celebrates its 20th anniversary with this special edition. Silverstein's humorous and creative verse can amuse the dowdiest of readers. Lemon
-faced adults and fidgety kids sit still and read these rhythmic words and laugh and smile and love th It's hard to imagine a world without A Light in the Attic. This now-class
ic collection of poetry and drawings from Shel Silverstein celebrates its 20th anniversary with this special edition. Silverstein's humorous and creative verse can amuse the do
wdiest of readers. Lemon-faced adults and fidgety kids sit still and read these rhythmic words and laugh and smile and love that Silverstein. Need proof of his genius? Rockabye
Rockabye baby, in the treetopDon't you know a treetopIs no safe place to rock?And who put you up there,And your cradle, too?Baby, I think someone down here'sGot it in for you.
Shel, you never sounded so good. ...more"}$
(.venv) → Jour3 git:(main) ×
```

Exercice 3 : Récupérer une liste de contenu spécifique

Prototype : *get_page_books*(url: str | None=None) -> list[dict]
Rendu : partie4.py

En continuant l'analyse de l'exercice précédent, trouver le pattern d'éléments HTML qui se répète pour chaque livre dans l'objectif de récupérer une liste de livres automatiquement.

Écrire une fonction *get_first_page* qui récupère le titre, note, prix et description de tous les livres affichés sur la page d'accueil. Retourner les informations dans une liste de dictionnaires, chaque dictionnaire ayant la même structure celui de l'exercice précédent.
Ajouter un paramètre optionnel `url`, qui peut contenir l'url d'une page ultérieure comme la page 5. Si ce paramètre n'est pas vide, utiliser cette URL à la place de celle de base.

Attention : Les liens entre la page d'accueil et les pages suivantes sont légèrement différents. Testez bien votre fonction !

Exercice 4 : Gestion de la pagination

Prototype : *get_page_range*(start: int, end: int) -> pd.DataFrame
Rendu : partie4.py

Maintenant qu'une liste de livres peut être récupérées, si on observe bien le site, on peut remarquer que seul 20 livres sur 1000 sont affichés sur la première page et qu'on peut changer de page pour afficher les 20 suivants. Continuer votre analyse de la structure du site pour trouver le pattern (HTML ou autre) permettant de changer de page.

Écrire une fonction *get_page_range* qui prend en paramètre une page de début et une page de fin (inclus) et récupère le titre, note, prix et description de tous les livres affichés dans la plage de pages prise en paramètre. Retourner les informations dans un DataFrame contenant 4 colonnes : `title`, `rating`, `price` et `description`.

Exemple :

```
df = get_page_range(1, 3)
print(df.tail(10))
```

```
(.venv) →  Jour3 git:(main) × python3 partie4.py | cat -e
Loading HTML from the cache$
Loading HTML from the cache$
```

...

```
Loading HTML from the cache$
                                          title  ...                                        description$
50  Unbound: How Eight Technologies Made Us Human,...  ...  Although we usually think of technology as som...$
51  Tsubasa: WoRLD CHRoNiCLE 2 (Tsubasa WoRLD CHRo...  ...  DUAL WORLDS, DUAL SIGHTS In the land of Nirai ...$
52  Throwing Rocks at the Google Bus: How Growth B...  ...  Capital in the Twenty-First Century meets The ...$
53                                  This One Summer  ...  Every summer, Rose goes with her mom and dad t...$
54                                           Thirst  ...  On a searing summer Friday, Eddie Chapman has ...$
55       The Torch Is Passed: A Harding Family Story  ...  Andrea Harding is a recent college graduate lo...$
56                  The Secret of Dreadwillow Carse  ...  A princess and a peasant girl must embark on a...$
57  The Pioneer Woman Cooks: Dinnertime: Comfort C...  ...  THEREâ€M-^@€M-^YS NO TIME LIKE DINNERTIME!Oh, donâ€M-^@€M-^Yt ...$
58                              The Past Never Ends  ...  A simple task, Attorney Chester Morgan thinks....$
59  The Natural History of Us (The Fine Art of Pre...  ...  One class assignment. One second chance at lov...$
$
[10 rows x 4 columns]$
(.venv) →  Jour3 git:(main) ×
```

Pour aller plus loin (optionnel) : Ajouter un paramètre optionnel « url » à votre fonction. Celui-ci peut contenir une URL d'accueil d'une catégorie (Nonfiction par exemple). Modifier votre fonction pour pouvoir récupérer tous les livres (pagination comprise) de la catégorie donnée. L'architecture des pages de catégories n'est pas exactement la même que celle de la page d'accueil du site, observez bien les différences.

Exercice 5 : Extraction de données par catégorie

Prototype : *get_category*(category: str) -> pd.DataFrame
Rendu : partie4.py

Dans la continuité des exercices précédents, rechercher où sont stockés les liens vers les différentes catégories dans la structure HTML du site.

Ensuite, écrire une fonction *get_category* qui prend en paramètre un nom de catégorie. Récupérer et retourner les informations de tous les livres de la catégorie donnée si elle existe, sinon retourner None. Les informations doivent être stockées dans un DataFrame contenant 3 colonnes : `title`, `rating` et `price`. Vous devez prendre en compte la pagination dans la catégorie si nécessaire.

Exemple :

```
df = get_category("Fiction")
print(df.tail(10))
```

```
(.venv) →  Jour3 git:(main) × python3 partie4.py | cat -e
Loading HTML from the cache$
Loading HTML from the cache$
Loading HTML from the cache$
Loading HTML from the cache$
Loading HTML from the cache$
                                        title  rating  price$
55          Jurassic Park (Jurassic Park #1)       1  44.97$
56               Inferno (Robert Langdon #4)       5  41.00$
57  Crazy Rich Asians (Crazy Rich Asians #1)       5  49.13$
58                           Big Little Lies       1  22.11$
59                     The Course of Love          3  16.78$
60                          When I'm Gone        3  51.96$
61                          The Silent Wife      5  12.34$
62                    The Bette Davis Club       3  30.66$
63            Kitchens of the Great Midwest      5  57.20$
64                              Bright Lines       5  39.07$
(.venv) →  Jour3 git:(main) ×
```

Exercice 6 : Stockage des informations et agrégation de masse

Prototype : *get_all_categories*(file: str) -> pd.DataFrame
Rendu : partie4.py

Écrire une fonction *get_all_categories* qui prend en paramètre un nom de fichier CSV. Récupérer et retourner les informations de tous les livres pour chaque catégorie. Vous devez prendre en compte la pagination dans la catégorie si nécessaire.
Les informations doivent être stockées dans un DataFrame avec la même structure que l'exercice précédent à laquelle il faut ajouter une colonne `category`. Enregistrer ce DataFrame (sans les index) dans le fichier CSV pris en paramètre.

Conseil : Stocker le DataFrame dans un fichier CSV permet de ne pas avoir à appeler le site à chaque analyse que vous allez effectuer dans les exercices suivants et donc à ne pas surcharger le site ou vous faire ban-ip de celui-ci.

Exercice 7 : Utilisation des données récupérées

Prototypes :

- *basic_statistics(*df: pd.DataFrame) -> (float, float)
- *categories_statistics*(df: pd.DataFrame) -> pd.DataFrame
- *price_distribution(df: pd.DataFrame) -> pd.DataFrame*

Rendu : partie4.py

Pour toutes les fonctions de cet exercice, la structure du DataFrame pris en paramètre sera la même que celui-ci de l'exercice précédent. Utiliser le fichier CSV précédemment créé pour ne pas avoir à surcharger le site d'appels inutiles.

Écrire une fonction *basic_statistics* qui prend un DataFrame. Retourner dans un tuple le prix moyen et la note moyenne de tous les livres.

Exemple :

```
df = pd.read_csv('resources/categories.csv')
price, rating = basic_statistics(df)
print(price, rating)
```

```
(.venv) →  Jour3 git:(main) ✗ python3 partie4.py | cat -e
35.07035 2.923$
(.venv) →  Jour3 git:(main) ✗
```

Écrire une fonction *categories_statistics* qui prend un DataFrame. Calculer pour chaque catégorie, le nombre de livre qu'elle contient ainsi que le prix moyen et la note moyenne. Retourner le résultat sous forme d'un DataFrame contenant 4 colonnes : `category`, `nb_book` et `avg_price ` et `avg_rating`. Vous devez arrondir le prix moyen et la note moyenne à la deuxième décimale.

Exemple :

```
df = pd.read_csv('resources/categories.csv')
df_categories = categories_statistics(df)
print(df_categories.tail(10))
```

```
(.venv) →  Jour3 git:(main) ✗ python3 partie4.py | cat -e
                nb_books  avg_price  avg_rating$
category                                      $
Suspense               1      58.33        3.00$
Thriller              11      31.43        2.73$
Travel                11      39.79        2.73$
Womens Fiction        17      36.79        3.12$
Young Adult           54      35.45        3.30$
(.venv) →  Jour3 git:(main) ✗
```

Écrire une fonction *price_distribution* qui prend un DataFrame. Calculer le nombre de livres par tranche de prix de 10£. Attention, les tranches doivent être dynamiques en fonction du prix des livres présent dans la DataFrame. Retourner le résultat sous forme d'un DataFrame contenant 2 colonnes : `range`, `nb_books`. La colonne `tranche` doit être un nombre entier correspondant au prix bas de chaque tranche (0, 10, 20, 30, …).

Indice : value_counts

Exemple :

```
df = pd.read_csv('resources/categories.csv')
df_prices = price_distribution(df)
print(df_prices)
```

```
(.venv) →  Jour3 git:(main) ✗ python3 partie4.py | cat -e
   range  nb_books$
0     10       196$
1     20       207$
2     30       194$
3     40       205$
4     50       198$
(.venv) →  Jour3 git:(main) ✗
```

Going Further : Mini-projet

Rendu : going_further.py

Pour cette section Going Further, nous vous proposons un mini-projet qui consolide votre apprentissage des deux derniers jours. Aujourd'hui, vous avez découvert les bibliothèques Python `requests` et `BeautifulSoup`, permettant d'effectuer facilement des tâches de scraping web. Il est maintenant temps d'appliquer ces compétences dans un scénario réel. Dans ce but, vous allez devoir extraire et analyser les informations de tous les députés français contenu sur le site VoxPublic.

Vous êtes libres dans la réalisation de ce projet et les analyses que vous souhaitez faire, voici cependant quelques idées et étapes à suivre :

- Analysez la structure du site dans l'objectif de récupérer les informations contenues sur celui-ci. Pour chaque député, vous pouvez par exemple récupérer les informations suivantes : nom, prénom, âge, profession, circonscription et groupe politique. Pensez à prendre en compte la pagination du site,
- Stocker le résultat de votre scraping dans un DataFrame. **Conseil :** Stocker le DataFrame dans un fichier CSV permet de ne pas avoir à appeler le site à chaque lancement de votre script et donc à ne pas surcharger le site ou vous faire ban de celui-ci,
- Idées d'analyses :
    - Calculer l'âge moyen des députés,
    - Regrouper les députés par département et compter leur nombre (Attention, la circonscription ne contient pas uniquement le nom du département, vous devez bien gérer la comparaison),
    - Compter pour chaque groupe politique le nombre de députés qu'il contient ainsi que l'âge moyen. Calculer uniquement les statistiques pour les députés ayant un groupe politique assigné,
    - Calculer pour chaque profession différente le nombre de députés ayant cette profession ainsi que l'âge moyen.