🕸️ TP : Scraper des articles sur *francetvinfo.fr* (BeautifulSoup, Python)

Dans ce TP, vous allez découvrir comment extraire automatiquement des informations à partir d'une page web grâce à **BeautifulSoup** en Python. L'objectif est de récupérer les titres, liens et dates des articles récents sur *francetvinfo.fr* puis de les sauvegarder dans un fichier JSON. Vous apprendrez ainsi les bases du **web scraping** : téléchargement de page, analyse du HTML et structuration des données.

Pré-requis

```
pip install requests beautifulsoup4
```

Étape 1 — Télécharger la page

Dans cette première étape du TP, nous apprenons à **récupérer le code HTML d'une page web**. Nous utilisons la bibliothèque `requests` pour envoyer une requête HTTP vers l'adresse `https://www.francetvinfo.fr/`. Si la réponse est correcte (code 200), nous affichons une partie du contenu de la page. Cela permet de vérifier que nous avons bien téléchargé le HTML avant de passer à l'analyse avec BeautifulSoup.

```
import requests  # Bibliothèque pour envoyer des requêtes HTTP

# Étape 1 : définir l'URL de la page à télécharger
URL = "https://www.francetvinfo.fr/"

# Étape 2 : envoyer une requête HTTP GET pour récupérer le contenu de la page
# On ajoute un en-tête "User-Agent" pour que le site nous traite comme un vrai navigateur
resp = requests.get(URL, headers={"User-Agent": "Mozilla/5.0"})

# Étape 3 : vérifier si la requête a réussi
if resp.status_code == 200:
    # Si le serveur a répondu "200 OK", on affiche un message
    print("Succès ! Contenu de la page récupéré.")
    # On affiche les 500 premiers caractères du code HTML pour voir à quoi cela ressemble
    print(resp.text[:500])
else:
    # Si la réponse n'est pas 200, on affiche le code d'erreur (ex : 404, 500…)
    print(f"Erreur : statut {resp.status_code}")
```

Étape 2 — Télécharger la page et l'enregistrer localement

À cette étape, l'objectif est de **sauvegarder la page HTML dans le répertoire /scrap** plutôt que d'afficher seulement quelques caractères dans la console. 💾 Ce code doit être placé **à la place du `print`** qui affichait les 500 premiers caractères, et il doit s'exécuter **uniquement si le code HTTP est bien 200** (c'est-à-dire que la requête a réussi).

Cela permettra d’avoir une copie locale du HTML (par exemple `page_francetvinfo.html`) que vous pourrez ensuite analyser tranquillement sans avoir besoin de recharger le site à chaque test.

```
import os  # à ajouter en haut du fichier

# S'assurer que le répertoire "scrap" existe, sinon le créer
os.makedirs("./scrap", exist_ok=True)

# Nouveau morceau de code pour écrire le HTML dans un fichier local
with open("./scrap/page_francetvinfo.html", "w", encoding="utf-8") as f:
    f.write(resp.text)
```

Etape 3 : commencer à explorer la page avec BeautifulSoup.

```
from bs4 import BeautifulSoup  # à ajouter en haut du fichier

# Étape 1 : créer un objet BeautifulSoup à partir du HTML téléchargé
soup = BeautifulSoup(resp.text, "html.parser")

# Étape 2 : sélectionner tous les éléments <h2> de la page
h2_elements = soup.find_all("h2")

# Étape 3 : stocker ces <h2> dans une liste
h2_list = [h.get_text(strip=True) for h in h2_elements]

# Étape 4 : afficher combien d'éléments on a trouvé
print("Nombre de balises <h2> :", len(h2_list))
```

Cet exemple montre comment :

1. **Analyser** le HTML téléchargé avec `BeautifulSoup`.
2. **Chercher toutes les balises** `<h2>` grâce à `find_all("h2")`.
3. **Stocker les titres** dans une liste Python.
4. **Compter** combien il y a d’éléments avec `len()`.

 Testez ce bout de code tel quel. Ensuite, complétez-le pour construire un **dictionnaire** contenant :

- les textes des `<h2>` comme clés,
- les liens (`href`) de la classe `card-article-majeure__link` comme valeurs.

Enfin, affichez ce dictionnaire.

Structure html à crapper :

```
<article data-cy="card-article-majeure" class="card-article-majeure ">
  <a
    href="/politique/gouvernement-de-sebastien-lecornu/direct-gouvernement-de-
sebastien-lecornu-les-socialistes-posent-leurs-conditions-face-aux-offres-de-dialogue-
du-premier-ministre_7493083.html"
    class="card-article-majeure__link" >
      <div class="card-article-majeure__img-wrapper">
      <div class="img-stamp-wrapper">
```

```
        <picture class="picture-wrapper">
            <source type="image/avif" srcset="https://www.franceinfo.fr/pictures..."
sizes="(min-width: 1200px) 466px, (min-width: 880px) 60vw, 90vw"/>
            <source srcset="https://www.franceinfo.fr/pictures..." sizes="(min-width:
1200px) 466px, (min-width: 880px) 60vw, 90vw"/>
            <img class="card-article-majeure__img" badgeLabel="Direct"
modifierClass="" sizes="(min-width: 1200px) 466px, (min-width: 880px) 60vw, 90vw"
data-placeholder-on-error="true" aria-hidden="true" alt=" (XOSE BOUZAS / HANS LUCAS /
AFP)" width="432" height="297" src="https://www.franceinfo.fr/pictures..."
fetchPriority="high" />
            </picture>

        <span data-cy="badge" class="badge badge--dark badge--medium badge--live"
aria-hidden="false" > </span>
    </div>
    </div>
    <div class="card-article-majeure__text-wrapper">
    <h2 class="card-article-majeure__title">Gouvernement de Sébastien Lecornu : le
Parti socialiste sera reçu mercredi matin par le Premier ministre, annonce Olivier
Faure</h2>

    <p class="card-article-majeure__chapo">Sébastien Lecornu entame une nouvelle
semaine de consultations, avec la CGT lundi et le PS et le PCF mercredi.</p>
   </div>
 </a>
...
</article>
```

Etape 5 : apprendre à reconstituer les URL complètes

```python
from urllib.parse import urljoin  # à ajouter en haut du fichier

BASE_URL = "https://www.francetvinfo.fr/"

# Exemple : si vous avez extrait un lien relatif "/politique/article.html"
relative_link = "/politique/article.html"

# urljoin permet de reconstituer l'URL absolue
full_url = urljoin(BASE_URL, relative_link)

print(full_url)  # affiche : https://www.francetvinfo.fr/politique/article.html
```

À cette étape, vous savez extraire les titres et les liens relatifs. Il faut maintenant **revoir la structure de votre dictionnaire** :

- **Clé** : le lien complet de l’article
- **Valeur** : un tuple `(titre, article)`

```
{
    "https://www.francetvinfo.fr/politique/article.html": ("Titre de
l’article", "Texte de l’article"),
```

```
    ...
}
```

Ensuite, pour chaque lien :

1. Téléchargez la page correspondante avec `requests`.
2. Analysez le HTML avec `BeautifulSoup`.
3. Extrayez le **titre complet** et le **texte principal de l'article**.

Etape 6 - factoriser le code

L'étape suivante — **factoriser le code** — consiste à découper ce gros bloc en **fonctions réutilisables** pour le rendre plus clair et maintenable.

1. **`fetch_html(url: str) -> str`**
   - Envoie une requête HTTP avec un User-Agent.
   - Retourne le contenu HTML sous forme de texte.
2. **`save_html(content: str, filepath: str) -> None`**
   - Vérifie que le dossier cible existe.
   - Écrit le contenu HTML dans un fichier local.
3. **`parse_main_page(html: str, classe: str) -> list`**
   - Crée un objet `BeautifulSoup`.
   - Sélectionne tous les éléments de la classe demandée.
   - Retourne la liste des liens relatifs.
4. **`scrape_article(url: str) -> tuple[str, str]`**
   - Télécharge la page d'un article.
   - Extrait le titre (`h1.c-title`) et le texte (`div.c-body`).
   - Retourne un tuple `(titre, article)`.
5. **`scrape_all_articles(base_url: str, hrefs: list) -> dict`**
   - Parcourt tous les liens d'articles.
   - Pour chacun, appelle `scrape_article`.
   - Construit un dictionnaire avec **clé = titre** et **valeur = (titre, article)**.
6. **`save_json(data: dict, filepath: str) -> None`**
   - Sauvegarde un dictionnaire Python dans un fichier JSON lisible.
7. **`main() -> None`**
   - Fonction principale qui appelle les étapes dans le bon ordre :
     1. Télécharger la page principale.
     2. Sauvegarder le HTML.
     3. Extraire les liens.
     4. Scraper chaque article.
     5. Sauvegarder en JSON.

Réécrire le script final en appelant uniquement `main()`.

Etape 7 — Récupérer *tous* les articles (quelle que soit la carte)

La page contient plusieurs formats de cartes, donc plusieurs classes de liens d’articles :

- `card-article-majeure__link`
- `card-article-related__link`
- `card-article-m__link` (…et potentiellement d’autres variantes)

Objectif : modifier votre extraction pour **récupérer tous les liens d’articles**, peu importe la classe utilisée.

- Vous récupérez les liens issus **de toutes les variantes de cartes** présentes.
- Les URLs sont **reconstituées** (absolues).
- Pas de doublons dans la collection finale.
- Votre pipeline enchaîne bien : **(lien)** → **(titre, article)**.
- Export JSON final OK.