

# **INTRODUCTION AU WEB SCRAPING**

## DÉFINITION DU WEB SCRAPING

Le Web Scraping est une technique utilisée pour extraire des données de sites Web. Cela se fait par programmation où un script simule la navigation d'un utilisateur sur le Web. Les données extraites peuvent être enregistrées dans une base de données ou un fichier.

## **APPLICATIONS DU WEB SCRAPING**

- Collecte de données pour l'analyse de marché.
- Surveillance des prix pour le commerce électronique.
- Agrégation de contenu pour le référencement.
- Extraction de données pour la recherche académique.
- Génération de leads pour le marketing.

# OUTILS ET BIBLIOTHÈQUES EN PYTHON POUR LE WEB SCRAPING

- **BeautifulSoup** : Analyse et extraction de données HTML.
- **Scrapy** : Cadre de scraping et d'indexation de sites Web.
- **Requests** : Envoi de requêtes HTTP pour obtenir des pages Web.
- **Selenium** : Automatisation des navigateurs pour les sites JavaScript dynamiques.
- **Pandas** : Analyse et manipulation de données extraites.

## LIMITATIONS ET ASPECTS LÉGAUX DU WEB SCRAPING

- **Performance** : Le scraping peut être lent et consommer beaucoup de ressources.
- **Blocs** : Les sites peuvent bloquer les scrapers via CAPTCHA ou IP ban.
- **Légalité** : Le scraping peut violer les conditions d'utilisation ou le droit d'auteur.
- **Éthique** : Collecter des données sans consentement peut être considéré comme non éthique.

# PRÉREQUIS TECHNIQUES POUR LE WEB SCRAPING

- Connaissance de base du langage Python.
- Compréhension du HTML et éventuellement du CSS.
- Savoir utiliser les requêtes HTTP et interpréter les réponses.
- Connaissance des expressions régulières pour le filtrage des données.
- Familiarité avec les outils de développement web pour inspecter les éléments du site.

# **COMPRENDRE LE HTML**

# STRUCTURE DE BASE D'UNE PAGE HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Titre de la page</title>
</head>
<body>
    Contenu de la page
</body>
</html>
```

- `<!DOCTYPE html>` déclare le type de document.
- `<html>` racine du document HTML.
- `<head>` contient les métadonnées.
- `<title>` titre de la page web.
- `<body>` corps de la page, contenu visible.

# BALISES HTML

- Balises de structure: <html>, <head>, <body>
- Balises de texte: <h1>, <p>, <span>
- Balises de lien: <a>
- Balises de liste: <ul>, <ol>, <li>
- Balises de tableau: <table>, <tr>, <td>

Balises auto-fermantes: <img />, <br />

## ATTRIBUTS DES BALISES

- Attributs communs: id, class, style
- <a href="url">: attribut href pour les liens
- : src et alt pour les images
- <input type="text" name="nom">: type et name pour les champs de formulaire

Les attributs définissent les propriétés des balises.

# CONTENU TEXTUEL ET MULTIMÉDIA

- Texte: <p>, <h1> à <h6>, <strong>, <em>
- Images: 
- Audio: <audio controls><source src="audio.mp3" type="audio/mpeg"></audio>
- Vidéo: <video controls><source src="video.mp4" type="video/mp4"></video>

Utilisation de balises pour intégrer différents types de contenu.



## LISTES

- Listes non ordonnées: <ul><li>Élément</li></ul>
- Listes ordonnées: <ol><li>Élément</li></ol>
- Description: <dl><dt>Terme</dt><dd>Description</dd></dl>

## TABLEAUX

```
<table>
  <tr>
    <th>En-tête</th>
  </tr>
  <tr>
    <td>Donnée</td>
  </tr>
</table>
```

- `<table>` pour créer un tableau.
- `<tr>` ligne du tableau.
- `<th>` cellule d'en-tête.
- `<td>` cellule standard.

# LIENS HYPERTEXTES

```
<a href="url">Texte du lien</a>
```

- `href` spécifie l'URL de la page cible.
- Peut lier vers des pages externes ou internes.
- Attribut `target="_blank"` pour ouvrir dans un nouvel onglet.
- Utilisation des ancrés avec `href="#ancre"` pour naviguer dans la même page.

# FORMULAIRES HTML

```
<form action="traitement.php" method="post">
  <label for="nom">Nom :</label>
  <input type="text" id="nom" name="nom">
  <input type="submit" value="Envoyer">
</form>
```

- <form> conteneur du formulaire.
- action URL de traitement des données.
- method méthode d'envoi (get ou post).
- <label> associe un texte à un champ.
- <input> différents types de champs (texte, bouton, etc.).

# **COMPRENDRE LE CSS**

## **NOTION DE CSS (CASCADING STYLE SHEETS)**

- CSS signifie Cascading Style Sheets.
- Utilisé pour le design et la mise en forme des pages web.
- Permet de séparer le contenu (HTML) de la présentation.
- Les règles CSS peuvent être incluses dans le HTML ou dans des fichiers externes.
- CSS contrôle la couleur, la police, l'espacement, la taille et bien plus.

## STRUCTURE D'UNE RÈGLE CSS

- Composée d'un sélecteur et d'un bloc de déclaration.
- Le sélecteur cible l'élément HTML à styliser.
- Le bloc de déclaration contient une ou plusieurs déclarations séparées par des points-virgules.
- Chaque déclaration inclut une propriété et une valeur, séparées par un deux-points.

```
sélecteur {  
    propriété: valeur;  
}
```

# SÉLECTEURS CSS

- Sélecteurs d'éléments : cible les balises HTML (`div`, `p`, `h1...`).
- Sélecteurs de classe : cible les éléments avec un attribut `class` spécifique (`.maClasse`).
- Sélecteurs d'ID : cible les éléments avec un attribut `id` spécifique (`#monId`).
- Sélecteurs d'attributs : cible les éléments avec un attribut donné (`[type="text"]`).

# PROPRIÉTÉS ET VALEURS CSS

- Les propriétés CSS définissent quel aspect de l'élément est stylisé (ex: color, font-size).
- Les valeurs CSS spécifient le style à appliquer (ex: red, 12px).
- Exemple de propriétés : background, width, height, margin, padding.

```
h1 {  
    color: blue;  
    font-size: 14px;  
}
```

# APPLICATION DU CSS SUR LE HTML

- CSS en ligne : style directement dans l'élément HTML via l'attribut `style`.
- CSS interne : bloc `<style>` dans l'en-tête `<head>` du document HTML.
- CSS externe : fichier séparé lié au HTML avec l'élément `<link>`.

```
<!-- Exemple CSS externe -->
<link rel="stylesheet" href="styles.css">
```

# LES CLASSES ET ID EN CSS

- Classe : attribut `class` utilisé pour grouper des éléments similaires.
- ID : attribut `id` unique pour identifier un élément spécifique.
- Les classes permettent de réutiliser des styles, tandis que les ID sont spécifiques.

```
/* Classe */
.maClasse {
  color: green;
}

/* ID */
#monId {
  color: red;
}
```

# LES PSEUDO-CLASSES ET PSEUDO-ÉLÉMENTS

- Pseudo-classes : cible l'état d'un élément (`:hover`, `:active`, `:focus`...).
- Pseudo-éléments : cible une partie spécifique d'un élément (`::before`, `::after`).

```
/* Pseudo-classe */
a:hover {
    color: orange;
}

/* Pseudo-élément */
p::first-line {
    font-weight: bold;
}
```

## BOX MODEL EN CSS

- Chaque élément HTML est considéré comme une boîte.
- Comprend margin, border, padding, et content.
- Permet de contrôler l'espacement et la taille des éléments.

```
.box {  
    margin: 10px;  
    border: 1px solid black;  
    padding: 20px;  
    width: 300px;  
}
```

# LA MISE EN PAGE AVEC CSS (FLEXBOX, GRID)

- Flexbox : modèle de boîte flexible pour la mise en page d'une interface utilisateur.
- Grid : permet de créer des mises en page en deux dimensions avec lignes et colonnes.

```
/* Flexbox */
.container {
  display: flex;
  justify-content: space-between;
}

/* Grid */
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
}
```

# **UTILISATION DES OUTILS DE DÉVELOPPEMENT WEB**

# OUVRIR LES OUTILS DE DÉVELOPPEMENT WEB

Pour ouvrir les outils de développement dans les navigateurs :

- **Chrome ou Edge:** Ctrl + Shift + I ou F12
- **Firefox:** Ctrl + Shift + C ou F12
- **Safari:** Command + Option + I Ces outils aident à analyser et à comprendre le code des pages web.

## **INSPECTER LES ÉLÉMENTS DU DOM**

1. Clic droit sur l'élément à inspecter.
2. Sélectionner "Inspecter" ou "Examiner l'élément".
3. Le panneau Elements/DOM s'ouvre avec l'élément sélectionné.
4. Explorez la structure HTML et les propriétés CSS.

## UTILISER LA CONSOLE JAVASCRIPT

- Accédez à l'onglet "Console" dans les outils de développement.
- Tapez des commandes JavaScript et appuyez sur Entrée.
- Utilisez `console.log()` pour afficher des informations.
- Idéal pour tester des scripts ou débuguer.

## ACCÉDER AUX RESSOURCES RÉSEAU

- Ouvrez l'onglet "Réseau" ou "Network".
- Rechargez la page pour voir les ressources se charger.
- Cliquez sur une ressource pour voir les détails.
- Analysez les en-têtes, réponses, et timings.

## **EXAMINER LES FEUILLES DE STYLE CSS**

- Dans l'onglet "Elements", trouvez le panneau "Styles".
- Visualisez et éditez les styles CSS appliqués à l'élément.
- Identifiez les styles hérités et calculés.
- Trouvez les fichiers CSS liés dans l'onglet "Sources".

## OBSERVER LES REQUÊTES HTTP

- L'onglet "Réseau" montre toutes les requêtes HTTP.
- Sélectionnez une requête pour voir la méthode, l'URL, les en-têtes et les réponses.
- Filtrez les requêtes par type (XHR, JS, CSS, etc.).
- Analysez la performance et les problèmes de chargement.

## TESTER ET MODIFIER LE DOM EN TEMPS RÉEL

- Dans l'onglet "Elements", cliquez sur un nœud DOM.
- Modifiez le HTML ou les styles et observez les changements en direct.
- Utilisez "Undo" (`Ctrl + Z`) pour annuler les modifications.
- Pratique pour tester des changements sans éditer le code source.

# **LES OUTILS DE WEB SCRAPING**

# BEAUTIFULSOUP

BeautifulSoup est une bibliothèque Python pour analyser des documents HTML et XML. Elle crée des parse trees qui sont utiles pour extraire des données facilement. Facile à utiliser et s'intègre bien avec les requêtes faites via `requests`.

Installation via pip:

```
pip install beautifulsoup4
```

Exemple basique:

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')
```

# REQUESTS

Requests est une bibliothèque HTTP pour Python. Elle permet d'envoyer des requêtes HTTP/1.1 avec une syntaxe simple. Utilisée pour obtenir le contenu d'une page web avant le parsing.

Installation via pip:

```
pip install requests
```

Exemple d'utilisation:

```
import requests
response = requests.get('http://example.com')
```

# SELENIUM

Selenium est un outil pour automatiser les navigateurs web. Permet de simuler l'interaction utilisateur pour le Web Scraping de sites dynamiques. Nécessite un driver de navigateur (ChromeDriver, GeckoDriver, etc.).

Installation via pip:

```
pip install selenium
```

Exemple d'initialisation:

```
from selenium import webdriver  
driver = webdriver.Chrome('/path/to/chromedriver')
```

# SCRAPY

Scrapy est un framework de Web Scraping et de crawling. Conçu pour le scraping à grande échelle et la collecte de données structurées. Permet de gérer les requêtes, le parsing et le stockage des données.

Installation via pip:

```
pip install scrapy
```

Exemple de commande:

```
scrapy startproject myproject
```

# LXML

LXML est une bibliothèque de traitement XML et HTML pour Python. Très rapide et facile à utiliser avec des API compatibles avec ElementTree. Supporte les requêtes XPath et XSLT.

Installation via pip:

```
pip install lxml
```

Exemple d'utilisation:

```
from lxml import etree
tree = etree.parse('page.html')
```

# MÉTHODES DE SÉLECTION DES ÉLÉMENTS

CSS Selectors et XPath sont deux méthodes pour sélectionner des éléments dans un document HTML.

CSS Selectors:

```
soup.select('div.class')
```

XPath:

```
tree.xpath('//div[@class="class"]')
```

Ces méthodes permettent de cibler des éléments spécifiques pour le scraping.

# INSTALLATION DES BIBLIOTHÈQUES DE WEB SCRAPING

Pour installer les bibliothèques de Web Scraping, utilisez pip:

```
pip install beautifulsoup4  
pip install requests  
pip install selenium  
pip install scrapy  
pip install lxml
```

Assurez-vous d'avoir Python et pip installés sur votre système.

# GESTION DES EXCEPTIONS EN WEB SCRAPING

La gestion des exceptions est cruciale pour un scraping fiable.

Exemples d'exceptions à gérer :

- `HTTPError` pour les problèmes de requête HTTP.
- `ElementNotVisibleException` pour les éléments non visibles avec Selenium.
- Gestion des timeouts et des erreurs de connexion.

Code d'exemple :

```
try:  
    response = requests.get(url)  
    response.raise_for_status()  
except requests.exceptions.HTTPError as e:  
    print(e)
```

# **LES BASES DU HTTP**

## **CONCEPTS FONDAMENTAUX DU HTTP**

Le protocole HTTP (Hypertext Transfer Protocol) est la base du web. Il permet la communication entre les clients web et les serveurs. Fonctionne selon un modèle de requête-réponse. Indépendant de la langue ou du type de données échangées. HTTP est un protocole sans état, mais non sans session.

# URL ET RESSOURCES WEB

URL (Uniform Resource Locator) identifie une ressource sur le web. Structure d'une URL:

http(s)://host:port/path?query#fragment

- http(s): Protocole utilisé
- host: Nom de domaine ou adresse IP
- port: Port de communication (optionnel)
- path: Chemin d'accès à la ressource
- query: Paramètres supplémentaires (optionnel)
- fragment: Section spécifique de la page (optionnel)

## **PROTOCOLE CLIENT-SERVEUR**

Modèle de communication entre deux programmes informatiques. Client envoie une requête HTTP au serveur. Serveur traite la requête et renvoie une réponse. Les rôles de client et serveur sont fixes dans une transaction. Exemple de clients: navigateurs web, applications de scraping.

# MÉTHODES HTTP DE BASE

Méthode	Description
GET	Demande de données (ressource)
POST	Soumission de données (créer/mettre à jour)
PUT	Remplacement complet d'une ressource
DELETE	Suppression d'une ressource
HEAD	Demande d'en-têtes sans le corps de réponse
OPTIONS	Demande d'informations sur la communication

# CODES DE STATUT HTTP

## Code Signification

---

200 OK : Réussite de la requête

---

301 Moved Permanently : Ressource déplacée

---

404 Not Found : Ressource non trouvée

---

500 Internal Server Error : Erreur serveur

---

503 Service Unavailable : Service indisponible

# EN-TÊTES HTTP

Les en-têtes HTTP fournissent des informations sur la requête ou la réponse. Types d'en-têtes :

- Généraux : Date, Cache-Control
- Requête : Host, User-Agent, Accept
- Réponse : Server, Content-Type, Content-Length
- Entités : Permettent de définir le corps de la requête/réponse

## **CORPS DE LA REQUÊTE ET DE LA RÉPONSE**

Le corps contient les données envoyées ou reçues. Dans une requête POST, le corps contient les données à soumettre. Dans une réponse, le corps contient la ressource demandée. Le type de contenu est spécifié par l'en-tête Content-Type.

## **SESSIONS ET COOKIES**

Les cookies permettent de maintenir un état dans les communications HTTP. Stockent des données côté client entre les requêtes. Les sessions utilisent les cookies pour identifier les utilisateurs. Les cookies sont envoyés via les en-têtes HTTP: Cookie pour la requête, Set-Cookie pour la réponse.

# LES MÉTHODES DE REQUÊTES HTTP

# GET

La méthode GET est utilisée pour demander des données à une ressource spécifique.

- Pas de corps de requête.
- Peut être mis en cache.
- Reste dans l'historique du navigateur.
- Peut être bookmarké.
- Limitations de longueur d'URL.

## POST

La méthode POST est utilisée pour envoyer des données à un serveur pour créer/mettre à jour une ressource.

- Les données sont incluses dans le corps de la requête.
- Pas de restrictions de taille de données.
- Ne peut pas être mis en cache ou bookmarké.
- N'apparaît pas dans l'historique du navigateur.

# HEAD

La méthode HEAD demande une réponse identique à celle d'une requête GET, mais sans le corps de la réponse.

- Utilisé pour récupérer les en-têtes.
- Peut être mis en cache.
- Aucun risque de compromettre les données.

## PUT

La méthode PUT remplace toutes les représentations actuelles de la ressource cible par les données de la requête.

- Utilisé pour mettre à jour des ressources existantes.
- Idempotent : plusieurs requêtes auront le même effet qu'une seule.
- Le corps de la requête contient la ressource mise à jour.

## **DELETE**

La méthode DELETE supprime la ressource spécifiée.

- Idempotent : plusieurs requêtes auront le même effet qu'une seule.
- Peut entraîner la suppression de données.
- Pas de corps de requête.

## PATCH

La méthode PATCH est utilisée pour appliquer des modifications partielles à une ressource.

- Seules les modifications sont envoyées.
- Moins lourd que PUT pour les mises à jour.
- Le corps de la requête contient les instructions de modification.

## OPTIONS

La méthode OPTIONS est utilisée pour décrire les options de communication pour la ressource cible.

- Renvoie les méthodes HTTP autorisées.
- Peut être utilisé pour vérifier les capacités du serveur.
- Pas de corps de requête.

# **ANALYSE DE LA STRUCTURE D'UN SITE WEB**

## **HTML ET LE DOM**

Le HTML (HyperText Markup Language) est le langage de balisage utilisé pour créer des pages web. Le DOM (Document Object Model) est une structure d'objet qui représente le document HTML dans le navigateur. Il permet aux langages de programmation d'interagir avec le contenu, la structure et le style de la page web.

## BALISES ET ATTRIBUTS HTML

Les balises HTML définissent les éléments de la page web, comme les paragraphes, les liens, les images, etc. Chaque balise peut avoir des attributs qui fournissent des informations supplémentaires ou modifient son comportement.

- Exemples de balises : `<p>`, `<a>`, `<img>`
- Exemples d'attributs : `href` pour les liens, `src` pour les images, `class` et `id` pour identifier les éléments.

## OUTILS D'INSPECTION DU NAVIGATEUR

Les navigateurs modernes offrent des outils d'inspection pour voir la structure HTML, le CSS et le JavaScript d'une page web. Pour accéder à ces outils :

- Faites un clic droit sur un élément de la page et sélectionnez "Inspecter".
- Utilisez les onglets "Éléments" et "Console" pour explorer le DOM et tester des scripts.

## **COMPRENDRE LE CSS POUR LE SCRAPING**

CSS (Cascading Style Sheets) est utilisé pour styliser les éléments HTML. Comprendre le CSS est essentiel pour le web scraping car il permet de sélectionner des éléments spécifiques via des sélecteurs.

- Sélecteurs de classe (.classe)
- Sélecteurs d'ID (#id)
- Sélecteurs d'attribut ([attribut=valeur])

## STRUCTURE DES URLs

L'URL (Uniform Resource Locator) est l'adresse d'une ressource sur Internet. Elle contient des informations importantes pour le scraping :

- Protocole (`http`, `https`)
- Domaine (`www.exemple.com`)
- Chemin (`/chemin/vers/la/page`)
- Paramètres de requête (`?cle=valeur`)

## **IDENTIFICATION DES DONNÉES PERTINENTES**

Pour scraper efficacement, il faut identifier les données pertinentes sur la page web :

- Repérer les éléments HTML contenant les données.
- Noter les classes ou ID uniques.
- Vérifier la présence de motifs récurrents dans le DOM pour les listes ou les tableaux de données.

# **EXTRACTION DE DONNÉES AVEC XPATH**

# INTRODUCTION À XPATH

XPath signifie XML Path Language. C'est un langage de requête pour sélectionner des nœuds dans un document XML. Utilisé aussi pour HTML avec des bibliothèques comme `lxml` en Python. Permet de naviguer dans l'arbre DOM et d'extraire des données. Syntaxe concise et puissante pour le Web Scraping.

## **SYNTAXE DE BASE D'XPATH**

- / Racine du document
- // Sélection de nœuds à n'importe quel niveau
- \* Sélection de tous les éléments enfants
- [@attribut='valeur'] Sélection par attribut
- nom\_noeud Sélection par nom de nœud
- [] Critères de sélection

# SÉLECTION D'ÉLÉMENTS AVEC XPATH

Pour sélectionner un élément spécifique :

- /html/body/div Sélectionne div dans body
- //div Sélectionne tous les div
- //div[@class='ma-classe'] Sélectionne div avec classe spécifique
- //div[1] Sélectionne le premier div

# SÉLECTION D'ATTRIBUTS AVEC XPATH

Pour sélectionner un attribut :

- //@href Sélectionne tous les attributs href
- //a/@href Sélectionne href des liens
- //img/@src Sélectionne src des images
- //div[@id='unique']/@style Style d'un div spécifique

## UTILISATION DES FONCTIONS XPATH

- `text ()` Récupère le texte d'un élément
- `contains (@attribut, 'texte')` Vérifie si attribut contient 'texte'
- `starts-with (@attribut, 'texte')` Vérifie le début de l'attribut
- `count (/div)` Compte le nombre de div
- `sum (/div/@data-value)` Somme des valeurs d'attributs

# NAVIGATION DANS L'ARBRE DOM AVEC XPATH

- . Nœud courant
- .. Nœud parent
- /div/span Enfants span du div
- //div//span Tous les span descendants de div
- //div/preceding-sibling::\* Frères précédents de div
- //div/following-sibling::\* Frères suivants de div

# EXTRACTION DE TEXTE AVEC XPATH

Pour extraire le texte :

- //p/text () Texte de tous les paragraphes
- //div[@class='description']/text () Texte des div avec classe spécifique
- string(//div[@id='unique']) Texte complet du div id unique
- //li[3]/text () Texte du troisième élément de la liste

# GESTION DES NAMESPACES AVEC XPATH

Les namespaces sont gérés par :

- Déclaration `xmlns:prefix='namespace-uri'`
- Utilisation `//prefix:element`
- Avec `lxml`, utiliser `{'prefix': 'namespace-uri'}`
- Ignorer les namespaces `//*[local-name()='element']`

# UTILISATION D'XPATH DANS PYTHON (AVEC LXML OU AUTRE BIBLIOTHÈQUE)

```
from lxml import etree

# Charger le document HTML
tree = etree.parse('page.html')

# Requête XPath
result = tree.xpath('//div[@class="content"]/text()')

# Afficher le résultat
print(result)
```

- Utiliser `etree.fromstring()` pour les chaînes XML/HTML.
- `xpath()` pour exécuter la requête XPath.
- Récupérer les résultats sous forme de liste.

# **EXTRACTION DE DONNÉES AVEC LES EXPRESSIONS RÉGULIÈRES**

## **COMPRENDRE LES EXPRESSIONS RÉGULIÈRES (REGEX)**

Les expressions régulières (Regex) sont des séquences de caractères qui forment un motif de recherche. Elles permettent d'effectuer des recherches complexes et des manipulations de chaînes de caractères. Utilisées pour valider, rechercher, extraire et remplacer du texte. Sont largement utilisées en programmation, y compris pour le Web Scraping.

## SYNTAXE DE BASE DES EXPRESSIONS RÉGULIÈRES

- . : correspond à n'importe quel caractère sauf une nouvelle ligne.
- ^ : correspond au début d'une chaîne de caractères.
- \$ : correspond à la fin d'une chaîne de caractères.
- \* : zéro ou plusieurs occurrences de l'élément précédent.
- + : une ou plusieurs occurrences de l'élément précédent.
- ? : zéro ou une occurrence de l'élément précédent.
- [ ] : correspond à n'importe quel caractère inclus entre les crochets.

# FONCTIONS PYTHON POUR LES EXPRESSIONS RÉGULIÈRES

- `re.match()` : vérifie si le Regex correspond au début de la chaîne.
- `re.search()` : recherche le Regex n'importe où dans la chaîne.
- `re.findall()` : trouve toutes les occurrences du Regex dans la chaîne.
- `re.sub()` : remplace les occurrences du Regex dans la chaîne.
- `re.compile()` : compile le Regex pour une utilisation répétée.



## **GESTION DES GROUPES DE CAPTURE DANS LES EXPRESSIONS RÉGULIÈRES**

- Les parenthèses () définissent un groupe de capture pour extraire des sous-chaînes.
- group (0) ou group () retourne la correspondance complète.
- group (n) retourne la n-ième sous-chaîne capturée.
- Les groupes nommés (?P<name> . . .) permettent d'extraire des sous-chaînes par leur nom.

## **CONSEILS POUR DES EXPRESSIONS RÉGULIÈRES EFFICACES ET OPTIMISÉES**

- Utiliser des classes de caractères spécifiques (\d, \w, \s) pour plus de précision.
- Préférer l'utilisation de quantificateurs non gourmands (\* ?, + ?) pour des recherches minimales.
- Compiler les Regex fréquemment utilisées avec `re.compile()` pour améliorer les performances.
- Utiliser les groupes de capture seulement quand nécessaire pour réduire la complexité.
- Tester et valider les Regex avec des outils en ligne pour éviter les erreurs.

# GESTION DE LA PAGINATION

## **COMPRÉHENSION DE LA PAGINATION**

La pagination est un processus qui divise le contenu en plusieurs pages. Cela permet de ne pas surcharger une seule page avec trop d'informations. Les sites web utilisent la pagination pour améliorer l'expérience utilisateur. Le WebScraping doit gérer la pagination pour extraire des données de toutes les pages.

# **IDENTIFICATION DES ÉLÉMENTS DE PAGINATION DANS LE HTML**

Les éléments de pagination sont généralement représentés par :

- Des boutons ou liens vers les pages suivantes ou précédentes.
- Un champ de numéro de page ou une liste déroulante. Examinez le code HTML pour trouver ces éléments :

```
<a href="page2.html">Suivant</a>
```

# UTILISATION DE BOUCLES POUR NAVIGUER ENTRE LES PAGES

Utilisez des boucles pour parcourir les pages :

```
while has_next_page:  
    # Code pour scraper la page  
    # Mettre à jour la condition has_next_page
```

Assurez-vous de gérer les temps d'attente pour ne pas surcharger le serveur.

# MISE À JOUR DES URLs POUR LES REQUÊTES SUCCESSIVES

Pour passer à la page suivante, modifiez l'URL :

```
base_url = "http://example.com/page="
page_num = 1
while page_num <= total_pages:
    url = base_url + str(page_num)
    # Code pour requête HTTP
    page_num += 1
```

## **GESTION DES PARAMÈTRES D'URL POUR LA PAGINATION**

Les paramètres d'URL peuvent inclure :

- page pour le numéro de la page.
- limit pour le nombre d'éléments par page.
- offset pour la position du premier élément sur la page. Exemple d'URL avec paramètres :

`http://example.com/?page=2&limit=10`

# **TECHNIQUES POUR ÉVITER LES PIÈGES DE LA PAGINATION INFINIE**

La pagination infinie charge de nouveaux contenus lorsque vous atteignez le bas de la page. Pour la gérer :

- Identifiez le déclencheur du chargement de nouveaux contenus (par exemple, un bouton "Plus").
- Simulez un clic ou extrayez l'URL de la prochaine charge de données si possible.
- Utilisez des outils comme Selenium pour contrôler un navigateur qui gère le JavaScript.

# **GESTION DES DONNÉES DYNAMIQUES AVEC JAVASCRIPT**

## **COMPRENDRE LE DOM (DOCUMENT OBJECT MODEL)**

Le DOM est une représentation structurée d'une page web. Il permet aux programmes d'interagir avec la structure de la page. Le DOM représente la page sous forme d'arbre d'objets. Chaque élément HTML est un nœud dans l'arbre DOM. JavaScript peut modifier le DOM pour mettre à jour la page.

## **UTILISATION DE SELENIUM**

Selenium est une bibliothèque pour automatiser les navigateurs. Elle permet de simuler des interactions utilisateur en Python. Installation via pip: `pip install selenium`. Nécessite un pilote de navigateur (ex: chromedriver pour Chrome). Permet d'accéder et de manipuler le DOM.

# INTERACTION AVEC LES ÉLÉMENTS DE LA PAGE

Avec Selenium, on peut:

- Trouver des éléments: `find_element_by_*`, `find_elements_by_*`.
- Cliquer sur des boutons: `element.click()`.
- Remplir des formulaires: `element.send_keys('texte')`.
- Récupérer du texte: `element.text`.
- Obtenir des attributs: `element.get_attribute('attribut')`.

## **GESTION DES ÉVÉNEMENTS JAVASCRIPT**

Les événements JavaScript sont des actions déclenchées sur la page. Selenium peut déclencher des événements comme le ferait un utilisateur. Exemple: passer la souris sur un élément, scroller, cliquer. Utilisation de ActionChains pour des séquences d'événements complexes.

## ATTENTE EXPLICITE ET IMPLICITE

Attente implicite: Selenium attend automatiquement la présence d'éléments.

- `driver.implicitly_wait(10)` attend jusqu'à 10 secondes. Attente explicite: attendre une condition spécifique avant de continuer.
- Utilisation de WebDriverWait avec `expected_conditions`.

## **SCRAPING DES DONNÉES GÉNÉRÉES PAR JAVASCRIPT**

Certaines données sont chargées dynamiquement par JavaScript. Selenium peut attendre que ces données soient chargées pour les scraper. Il est possible de détecter des changements dans le DOM. Les données dynamiques peuvent être récupérées après leur apparition.

## **UTILISATION D'APIS POUR RÉCUPÉRER DES DONNÉES DYNAMIQUES**

Certaines pages web utilisent des APIs pour charger des données dynamiques. Il est possible de faire des requêtes directement aux APIs pour récupérer les données. Utilisation de la bibliothèque `requests` pour interroger les APIs. Extraction des données à partir du JSON retourné par les APIs.

# ÉTHIQUE ET LÉGALITÉ DU WEB SCRAPING

## **RESPECT DES CONDITIONS GÉNÉRALES D'UTILISATION (CGU)**

- CGU définissent les règles d'utilisation d'un site web
- Toujours lire et respecter les CGU avant de scraper
- CGU peuvent interdire explicitement le scraping
- Ignorer les CGU peut entraîner des conséquences légales

## CONFORMITÉ AU ROBOTS.TXT

- Fichier robots.txt guide les robots d'indexation
- Indique les parties du site à ne pas visiter pour les robots
- Respecter robots.txt est une norme de courtoisie sur le web
- Scraper des zones interdites peut être perçu comme hostile

## **RESPECT DU DROIT D'AUTEUR ET DU CONTENU PROTÉGÉ**

- Contenus sur internet sont souvent soumis au droit d'auteur
- Ne pas copier ou redistribuer du contenu protégé sans autorisation
- Utiliser les données pour des analyses internes peut être acceptable
- Toujours vérifier la licence et demander des permissions si nécessaire

## **CONSIDÉRATIONS SUR LA CHARGE SERVEUR ET LE DÉNI DE SERVICE**

- Le scraping intensif peut surcharger les serveurs web
- Risque de ralentir ou rendre indisponible le service pour les autres
- Planifier le scraping pendant les heures creuses
- Limiter la fréquence des requêtes pour minimiser l'impact

# **CONFIDENTIALITÉ ET PROTECTION DES DONNÉES PERSONNELLES**

- Ne pas collecter de données personnelles sans consentement
- Respecter la vie privée des utilisateurs
- Être transparent sur l'utilisation des données collectées
- Se conformer aux lois sur la protection des données (ex: RGPD)

## **ASPECTS LÉGAUX SELON LES JURIDICTIONS ET LOIS APPLICABLES (EX: RGPD EN EUROPE)**

- Les lois varient d'un pays à l'autre
- RGPD en Europe protège les données personnelles
- Vérifier les lois locales sur la collecte de données
- Respecter les lois applicables pour éviter les sanctions

## BONNES PRATIQUES ET ÉTHIQUE DU SCRAPING

- Informer le site web de vos intentions de scraping
- Utiliser les API officielles si disponibles
- Ne pas nuire à l'expérience utilisateur du site
- Agir de manière responsable et éthique lors du scraping

# **STOCKAGE DES DONNÉES EXTRAITES**

# CHOIX DU FORMAT DE STOCKAGE

- Critères de choix :
  - Facilité d'accès et de manipulation
  - Compatibilité avec des outils d'analyse
  - Volume des données
  - Structure des données
- Formats courants :
  - Fichiers CSV
  - Bases de données (SQL, NoSQL)
  - Fichiers JSON
  - Fichiers XML

# UTILISATION DE FICHIERS CSV

- CSV : Valeurs séparées par des virgules
- Avantages :
  - Simplicité
  - Largement supporté
- Création en Python :

```
import csv
with open('data.csv', 'w', newline='') as file
    writer = csv.writer(file)
    writer.writerow(["header1", "header2"])
    writer.writerow(["value1", "value2"])
```

# UTILISATION DE BASES DE DONNÉES

- SQL (MySQL, PostgreSQL) vs NoSQL (MongoDB)
- Avantages :
  - Gestion de grandes quantités de données
  - Requêtes complexes
  - Sécurité
- Exemple en Python avec SQLite :

```
import sqlite3
conn = sqlite3.connect('data.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS table_name (id INTEGER PRIMARY KEY, col1 T
conn.commit()
conn.close()
```

# STOCKAGE EN JSON

- JSON : JavaScript Object Notation
- Avantages :
  - Lisible par l'homme
  - Facilement interopérable
- Création en Python :

```
import json
data = {"key": "value"}
with open('data.json', 'w') as json_file:
    json.dump(data, json_file)
```

# STOCKAGE EN XML

- XML : eXtensible Markup Language
- Avantages :
  - Structuration hiérarchique
  - Personnalisable
- Création en Python :

```
import xml.etree.ElementTree as ET
root = ET.Element("root")
child = ET.SubElement(root, "child")
child.text = "content"
tree = ET.ElementTree(root)
tree.write("data.xml")
```

# **UTILISATION DE BIBLIOTHÈQUES PYTHON POUR LE STOCKAGE**

- Bibliothèques populaires :
  - csv pour les fichiers CSV
  - sqlite3 pour les bases de données SQLite
  - json pour les fichiers JSON
  - xml.etree.ElementTree pour les fichiers XML
  - pandas pour la manipulation et le stockage de données

# GESTION DES ERREURS DE STOCKAGE

- Erreurs courantes :
  - Problèmes de permissions
  - Espace disque insuffisant
  - Formats de données incompatibles
- Bonnes pratiques :
  - Validation des données avant stockage
  - Gestion des exceptions
  - Logs des erreurs
  - Tests unitaires

# PERFORMANCE DU STOCKAGE DES DONNÉES

- Facteurs influençant la performance :
  - Taille des données
  - Fréquence d'accès
  - Type de stockage (disque dur, SSD)
- Optimisation :
  - Indexation des bases de données
  - Compression des données
  - Choix du format adapté à l'usage
- Monitoring :
  - Suivi de l'utilisation des ressources
  - Profilage des opérations de stockage