Formation Web Scraping avec Python

m2iformation.fr

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Web Scraping avec Python

Extraction automatique de données web

Objectifs

Compétences visées

- Comprendre la structure HTML et CSS des pages web
- Extraire des données structurées depuis des sites web
- Automatiser la collecte de données avec Python
- Gérer les aspects légaux et éthiques du scraping
- Créer des scrapers robustes et maintenables

Public

Data engineers, data analysts, développeurs Python, data scientists

Prérequis

Connaissances de base en Python (variables, fonctions, listes, dictionnaires)

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Fondamentaux du Web Scraping

Qu'est-ce que le Web Scraping ?

Définition :
Extraction automatique de données depuis des sites web.
Cas d'usage

1. **Veille concurrentielle**
   - Prix des produits
   - Disponibilité des stocks
   - Analyse de marché
2. **Agrégation de données**
   - Actualités
   - Offres d'emploi
   - Données immobilières
3. **Recherche et analyse**
   - Sentiment analysis sur réseaux sociaux
   - Études académiques
   - Data mining

Web Scraping vs API

API (Application Programming Interface)

- Données structurées (JSON/XML)
- Documentation officielle
- Stable et fiable
- Limites de taux (rate limits)
- Données limitées
- Nécessite authentification

Web Scraping

- Accès à toutes les données visibles
- Pas de limites officielles
- Pas d'inscription nécessaire
- Structure peut changer
- Moins stable
- Questions légales

Règle d'or : Privilégiez toujours l'API quand elle existe !

Aspects légaux et éthiques

Cadre légal

1. **robots.txt** : Fichier indiquant ce qui peut être scrapé
   - Exemple : `https://example.com/robots.txt`
   - À respecter impérativement
2. **Conditions d'utilisation (ToS)**
   - Lire les CGU du site
   - Certains sites interdisent explicitement le scraping
3. **Droit d'auteur et propriété intellectuelle**
   - Les données peuvent être protégées
   - Attention à l'utilisation commerciale
4. **RGPD (données personnelles)**
   - Ne pas scraper de données personnelles sensibles
   - Respecter le droit à l'oubli

Bonnes pratiques éthiques

Être un "bon citoyen" du web

1. **Rate limiting** : Ne pas surcharger le serveur
   - Ajouter des délais entre requêtes (1-2 secondes)
   - Scraper pendant les heures creuses
2. **User-Agent** : S'identifier correctement
   - Utiliser un User-Agent descriptif
   - Inclure un email de contact
3. **Politesse**
   - Respecter robots.txt
   - Arrêter si le site bloque
   - Ne pas contourner les protections
4. **Utilisation des données**
   - Ne pas revendre sans autorisation
   - Citer la source
   - Respecter la vie privée

robots.txt - Exemple

Fichier robots.txt

```
User-agent: *
Disallow: /admin/
Disallow: /private/
Disallow: /api/
Allow: /api/public/

User-agent: Googlebot
Allow: /

Crawl-delay: 10
```

Vérification en Python

```python
import requests
url = 'https://example.com/robots.txt'
response = requests.get(url)
print(response.text)
```

Ou utiliser urllib.robotparser

```python
from urllib.robotparser import RobotFileParser
rp = RobotFileParser()
rp.set_url("https://example.com/robots.txt")
rp.read()
print(rp.can_fetch("*", "https://example.com/page"))  # True/False
```

Structure HTML - Bases

HTML = HyperText Markup Language

Structure d'une page HTML :

```html
<!DOCTYPE html>
<html>
<head>
    <title>Titre de la page</title>
    <meta charset="UTF-8">
</head>
<body>
    <h1>Titre principal</h1>
    <p>Un paragraphe de texte.</p>
    <a href="https://example.com">Un lien</a>
    <img src="image.jpg" alt="Description">

    <div class="container">
        <ul>
            <li>Élément 1</li>
            <li>Élément 2</li>
        </ul>
    </div>
</body>
</html>
```

Balises HTML courantes

Balises de structure

- `<div>` : Conteneur générique
- `<span>` : Conteneur inline
- `<header>`, `<footer>`, `<nav>`, `<section>`, `<article>` : Sémantiques

Balises de texte

- `<h1>` à `<h6>` : Titres
- `<p>` : Paragraphe
- `<a>` : Lien (href)
- `<strong>`, `<em>` : Mise en forme

Balises de liste

- `<ul>` : Liste non ordonnée
- `<ol>` : Liste ordonnée
- `<li>` : Élément de liste

Balises de tableau

- `<table>`, `<tr>`, `<td>`, `<th>`

Attributs HTML

Attributs communs

```
<div id="unique-id" class="class1 class2" data-custom="value">
    Contenu
</div>

<a href="https://example.com" target="_blank" title="Infobulle">
    Lien
</a>

<img src="image.jpg" alt="Description" width="300" height="200">

<input type="text" name="username" placeholder="Entrez votre nom">
```

Attributs importants pour le scraping

- `id` : Identifiant unique
- `class` : Classes CSS (peut être multiple)
- `href` : URL d'un lien
- `src` : Source d'une image/script
- `data-*` : Attributs personnalisés

Sélecteurs CSS - Bases

CSS = Cascading Style Sheets

```css
/* Sélecteur de balise */
p { color: blue; }

/* Sélecteur de classe */
.ma-classe { font-size: 16px; }

/* Sélecteur d'ID */
#mon-id { background: yellow; }

/* Sélecteur d'attribut */
a[href="https://example.com"] { color: red; }

/* Sélecteurs combinés */
div p { }              /* p dans un div */
div > p { }            /* p enfant direct de div */
div + p { }            /* p juste après div */
p.ma-classe { }        /* p avec classe ma-classe */
```

Ces sélecteurs seront utilisés pour le scraping !

Inspection des pages web

Outils de développement (DevTools)
Chrome/Edge/Firefox : F12 ou Clic droit → "Inspecter"
Fonctionnalités utiles

1. **Inspecteur d'éléments**
   - Survolez un élément → voir le HTML
   - Clic droit → Copier → Sélecteur CSS
2. **Onglet Network**
   - Voir toutes les requêtes HTTP
   - Voir les headers, responses
   - Identifier les appels API
3. **Console**
   - Tester des sélecteurs
   - `document.querySelector('.ma-classe')`

Requêtes HTTP avec Requests

Récupérer une page web

```python
import requests

# Requête GET simple
url = 'https://example.com'
response = requests.get(url)

# Vérifier le statut
print(response.status_code)  # 200 = OK

# Contenu HTML
html = response.text
print(html[:500])  # Premiers 500 caractères

# Contenu binaire (images, PDF...)
content = response.content

# Encodage
print(response.encoding)  # utf-8, ISO-8859-1...

# Headers de la réponse
print(response.headers)
```

Headers et User-Agent

Pourquoi personnaliser les headers ?

- Certains sites bloquent les requêtes sans User-Agent
- S'identifier comme un navigateur
- Éviter d'être détecté comme un bot

```
import requests

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36',
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
    'Accept-Language': 'fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7',
    'Accept-Encoding': 'gzip, deflate',
    'Connection': 'keep-alive',
    'Referer': 'https://www.google.com/'
}

response = requests.get(url, headers=headers)
```

Bonnes pratiques

- Utiliser un User-Agent réel et à jour
- Inclure un email de contact (optionnel mais poli)
- Varier les User-Agents si scraping intensif

Gestion des erreurs

```python
import requests
from requests.exceptions import RequestException, Timeout, ConnectionError

def fetch_page(url, timeout=10):
    """Récupère une page avec gestion d'erreurs"""
    try:
        response = requests.get(
            url,
            headers=headers,
            timeout=timeout
        )
        response.raise_for_status()  # Lève exception si 4xx ou 5xx
        return response.text

    except Timeout:
        print(f"Timeout pour {url}")
        return None

    except ConnectionError:
        print(f"Erreur de connexion pour {url}")
        return None

    except requests.exceptions.HTTPError as e:
        print(f"Erreur HTTP {response.status_code}: {url}")
        return None

    except RequestException as e:
        print(f"Erreur générale: {e}")
        return None

# Utilisation
html = fetch_page('https://example.com')
if html:
    # Traiter le HTML
    pass
```

Sessions et cookies

Session = Maintenir l'état entre requêtes

```python
import requests

# Créer une session
session = requests.Session()

# Les cookies et headers sont persistés
session.headers.update({'User-Agent': 'My Scraper 1.0'})

# Première requête (authentification par exemple)
response1 = session.post(
    'https://example.com/login',
    data={'username': 'user', 'password': 'pass'}
)

# Requêtes suivantes gardent les cookies
response2 = session.get('https://example.com/dashboard')
response3 = session.get('https://example.com/profile')

# Voir les cookies
print(session.cookies.get_dict())

# Fermer la session
session.close()
```

Rate Limiting et politesse

Ne pas surcharger le serveur

```python
import time
import requests

urls = [
    'https://example.com/page1',
    'https://example.com/page2',
    'https://example.com/page3'
]

for url in urls:
    response = requests.get(url)
    print(f"Scraped: {url}")

    # Attendre entre chaque requête
    time.sleep(2)  # 2 secondes

# Ou utiliser random pour varier
import random
time.sleep(random.uniform(1, 3))  # Entre 1 et 3 secondes
```

Recommandations

- 1-2 secondes minimum entre requêtes
- Plus long pour sites lents ou fragiles
- Adapter selon le volume de requêtes

Sites d'entraînement

Sites spécialement conçus pour le scraping

1. **http://quotes.toscrape.com**
   - Citations célèbres
   - Plusieurs pages
   - Différentes structures HTML
2. **http://books.toscrape.com**
   - Catalogue de livres
   - Prix, notes, catégories
   - Pagination
3. **https://scrapethissite.com**
   - Exercices progressifs
   - Différents challenges

À éviter pour l'apprentissage

- Sites e-commerce réels
- Sites protégés par Cloudflare
- Sites avec authentification

Extraction avec BeautifulSoup

BeautifulSoup - Introduction

Qu'est-ce que BeautifulSoup ?

- Bibliothèque Python pour parser et naviguer dans le HTML/XML
- Transforme le HTML en arbre d'objets Python
- Facile à utiliser et très populaire

Installation

```
pip install beautifulsoup4 lxml
```

Import

```
from bs4 import BeautifulSoup
import requests

# Récupérer la page
response = requests.get('https://example.com')

# Parser le HTML
soup = BeautifulSoup(response.text, 'lxml')
# ou 'html.parser' (parser par défaut de Python)
```

Premier parsing

```
from bs4 import BeautifulSoup

html = """
<html>
<head><title>Ma Page</title></head>
<body>
    <h1 class="main-title">Titre Principal</h1>
    <p class="intro">Premier paragraphe.</p>
    <p>Deuxième paragraphe.</p>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>
    <a href="https://example.com" class="external">Lien</a>
</body>
</html>
"""

soup = BeautifulSoup(html, 'lxml')

# Accès direct aux balises
print(soup.title)        # <title>Ma Page</title>
print(soup.title.string) # Ma Page
print(soup.h1)           # Première balise h1
print(soup.p)            # Premier p

# Pretty print
print(soup.prettify())
```

find() et find_all()

find() : Trouve le premier élément

```
# Par nom de balise
first_p = soup.find('p')

# Par classe
intro = soup.find('p', class_='intro')

# Par ID
header = soup.find(id='header')

# Par attribut
link = soup.find('a', href='https://example.com')

# Avec attrs
div = soup.find('div', attrs={'data-id': '123'})
```

find_all() : Trouve tous les éléments

```
# Tous les paragraphes
all_p = soup.find_all('p')

# Tous les éléments avec une classe
intros = soup.find_all(class_='intro')

# Plusieurs balises
elements = soup.find_all(['p', 'div', 'span'])

# Limite de résultats
first_five = soup.find_all('li', limit=5)
```

Sélecteurs CSS avec select()

Plus puissant que find/find_all

```
# Sélecteur de classe
items = soup.select('.item')

# Sélecteur d'ID
header = soup.select('#header')

# Sélecteur de balise
paragraphs = soup.select('p')

# Combinaisons
div_paragraphs = soup.select('div p')      # p dans div
direct_children = soup.select('div > p')   # p enfant direct
class_and_tag = soup.select('p.intro')     # p avec classe intro

# Attributs
external_links = soup.select('a[href^="http"]')  # href commence par http
target_blank = soup.select('a[target="_blank"]')

# Pseudo-classes
first_item = soup.select('li:first-child')
nth_item = soup.select('li:nth-child(3)')
```

Extraction de texte

```
from bs4 import BeautifulSoup

html = """
<div class="article">
    <h2>Titre de l'article</h2>
    <p>Premier paragraphe avec <strong>texte en gras</strong>.</p>
    <p>Deuxième paragraphe.</p>
</div>
"""

soup = BeautifulSoup(html, 'lxml')

# .text ou .get_text() : Tout le texte
div = soup.find('div', class_='article')
print(div.text)
print(div.get_text())

# Avec séparateur
print(div.get_text(separator=' | '))

# Strip (supprimer espaces)
print(div.get_text(strip=True))

# .string : Texte direct (pas les enfants)
h2 = soup.find('h2')
print(h2.string)  # "Titre de l'article"

# .strings : Itérateur sur tous les textes
for text in div.strings:
    print(repr(text))
```

Extraction d'attributs

```
html = """
<a href="https://example.com" class="external" target="_blank" title="Example">
    Lien
</a>
<img src="image.jpg" alt="Mon image" width="300" data-id="img1">
"""

soup = BeautifulSoup(html, 'lxml')

# Accès direct avec []
link = soup.find('a')
href = link['href']
classes = link['class']  # Liste
title = link['title']

# get() pour éviter les erreurs
target = link.get('target')          # '_blank'
missing = link.get('missing', 'N/A')  # 'N/A' si absent

# Tous les attributs
print(link.attrs)  # Dictionnaire

# Image
img = soup.find('img')
src = img['src']
alt = img['alt']
data_id = img.get('data-id')
```

Navigation dans l'arbre DOM

```
html = """
<div class="parent">
    <h2>Titre</h2>
    <p>Paragraphe 1</p>
    <p>Paragraphe 2</p>
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>
</div>
"""

soup = BeautifulSoup(html, 'lxml')
div = soup.find('div', class_='parent')

# Enfants directs
children = list(div.children)
for child in children:
    if child.name:  # Ignorer les textes/espaces
        print(child.name)

# Tous les descendants
descendants = list(div.descendants)

# Parent
h2 = soup.find('h2')
print(h2.parent.name)  # 'div'

# Siblings (frères et sœurs)
p1 = soup.find('p')
print(p1.next_sibling.next_sibling)  # Prochain p (skip le \n)

# Méthodes utiles
next_p = p1.find_next_sibling('p')
prev = p1.find_previous_sibling()
```

Traitement de tableaux HTML

```
html = """
<table class="data">
    <thead>
        <tr>
            <th>Nom</th>
            <th>Age</th>
            <th>Ville</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Alice</td>
            <td>25</td>
            <td>Paris</td>
        </tr>
        <tr>
            <td>Bob</td>
            <td>30</td>
            <td>Lyon</td>
        </tr>
    </tbody>
</table>
"""

soup = BeautifulSoup(html, 'lxml')

# Méthode 1 : Manuelle
table = soup.find('table', class_='data')

# Headers
headers = [th.text for th in table.find('thead').find_all('th')]

# Lignes
rows = []
for tr in table.find('tbody').find_all('tr'):
    row = [td.text for td in tr.find_all('td')]
    rows.append(row)

# Créer DataFrame
import pandas as pd
df = pd.DataFrame(rows, columns=headers)

# Méthode 2 : Avec Pandas (plus simple)
df = pd.read_html(str(table))[0]
```

Extraction de liens

```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse

url = 'http://quotes.toscrape.com'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')

# Tous les liens
all_links = soup.find_all('a')

# Extraire href
hrefs = [link.get('href') for link in all_links if link.get('href')]

# Filtrer les liens externes
def is_external(link, base_domain):
    return urlparse(link).netloc and urlparse(link).netloc != base_domain

base_domain = urlparse(url).netloc
external_links = [link for link in hrefs if is_external(link, base_domain)]

# Construire URLs absolues
absolute_links = [urljoin(url, href) for href in hrefs]

# Liens uniques
unique_links = list(set(absolute_links))

print(f"Total liens : {len(all_links)}")
print(f"Liens externes : {len(external_links)}")
print(f"Liens uniques : {len(unique_links)}")
```

Extraction d'images

```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
import os

url = 'http://books.toscrape.com'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')

# Toutes les images
images = soup.find_all('img')

# Extraire src
image_urls = []
for img in images:
    src = img.get('src')
    alt = img.get('alt', 'No description')

    # URL absolue
    absolute_url = urljoin(url, src)

    image_urls.append({
        'src': absolute_url,
        'alt': alt
    })

# Télécharger une image
def download_image(image_url, save_path):
    response = requests.get(image_url)
    with open(save_path, 'wb') as f:
        f.write(response.content)

# Exemple
if image_urls:
    download_image(image_urls[0]['src'], 'data/output/image.jpg')
```

Nettoyage de données

```
from bs4 import BeautifulSoup
import re

html = """
<div>
    <p>  Texte avec   espaces  multiples  </p>
    <p>Prix: £51.77</p>
    <p>★★★★☆ (4/5)</p>
</div>
"""

soup = BeautifulSoup(html, 'lxml')

# 1. Nettoyer espaces
text = soup.find('p').get_text(strip=True)
text_clean = ' '.join(text.split())  # Remplace espaces multiples

# 2. Extraire prix
price_text = soup.find_all('p')[1].text
price = float(re.findall(r'[\d.]+', price_text)[0])

# 3. Extraire note
rating_text = soup.find_all('p')[2].text
rating = int(re.search(r'(\d)/5', rating_text).group(1))

# 4. Supprimer symboles
clean_text = re.sub(r'[★☆£]', '', text)

print(f"Texte nettoyé : {text_clean}")
print(f"Prix : {price}")
print(f"Note : {rating}/5")
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Automatisation avec Scrapy

Pourquoi Scrapy ?

Requests + BeautifulSoup vs Scrapy

Requests + BeautifulSoup

- Simple et rapide à apprendre
- Bon pour petits projets
- Gestion manuelle de la pagination
- Pas de parallélisation native
- Pas de gestion avancée

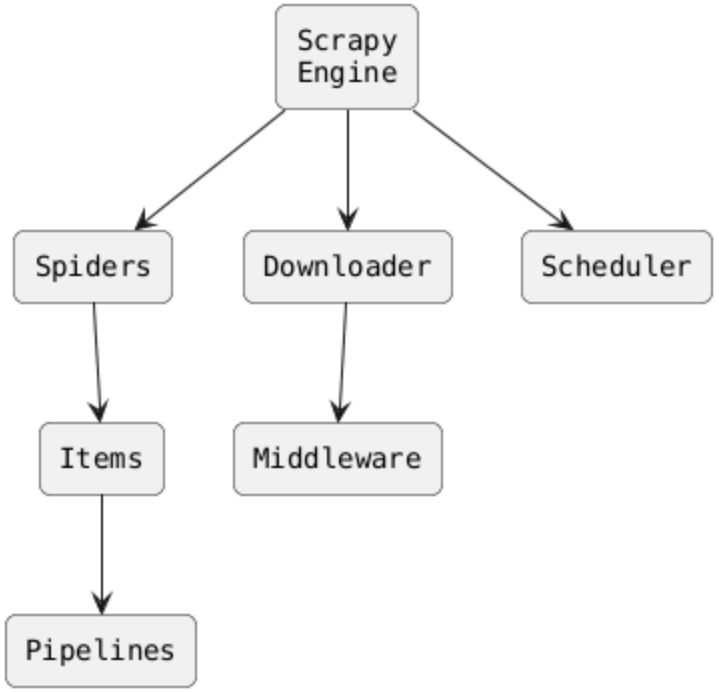
Scrapy ★

- Framework complet
- Parallélisation automatique (async)
- Gestion de pagination built-in
- Middleware et pipelines
- Retry automatique
- Export multiple formats
- Courbe d'apprentissage plus élevée

Quand utiliser Scrapy ?

- Projets moyens à grands
- Scraping régulier/production
- Besoin de performance

Architecture Scrapy



Composants

- **Engine** : Orchestrateur
- **Scheduler** : File d'attente des URLs
- **Downloader** : Télécharge les pages
- **Spiders** : Logique d'extraction
- **Items** : Données structurées
- **Pipelines** : Traitement des données

Installation et création de projet

Installation

```
pip install scrapy
```

Créer un projet

```
scrapy startproject myproject
cd myproject
```

Structure créée

```
myproject/
    scrapy.cfg              # Configuration du projet
    myproject/
        __init__.py
        items.py            # Définition des items
        middlewares.py      # Middlewares custom
        pipelines.py        # Pipelines de traitement
        settings.py         # Configuration
        spiders/            # Dossier des spiders
            __init__.py
```

Premier Spider

Créer un spider

```
scrapy genspider quotes quotes.toscrape.com
```

Fichier généré : spiders/quotes_spider.py

```
import scrapy

class QuotesSpider(scrapy.Spider):
    name = "quotes"  # Identifiant unique
    allowed_domains = ["quotes.toscrape.com"]
    start_urls = ["http://quotes.toscrape.com"]

    def parse(self, response):
        # Méthode appelée pour chaque réponse
        # response : objet contenant la page HTML

        # Extraire des données
        quotes = response.css('div.quote')

        for quote in quotes:
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get(),
                'tags': quote.css('a.tag::text').getall()
            }
```

Lancer le spider

```
scrapy crawl quotes
```

Selectors Scrapy

CSS Selectors (recommandé)

```
# Texte
response.css('span.text::text').get()       # Premier
response.css('span.text::text').getall()    # Tous

# Attribut
response.css('img::attr(src)').get()

# Nested
response.css('div.quote span.text::text').get()
```

XPath (plus puissant)

```
# Texte
response.xpath('//span[@class="text"]/text()').get()

# Attribut
response.xpath('//img/@src').get()

# Contains
response.xpath('//div[contains(@class, "quote")]')

# Parent
response.xpath('//span[@class="text"]/parent::div')
```

Méthodes

- `.get()` : Premier résultat (ou None)
- `.getall()` : Liste de tous les résultats
- `.re()` : Extraction avec regex

Pagination avec Scrapy

```python
import scrapy

class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = ["http://quotes.toscrape.com"]

    def parse(self, response):
        # Extraire données de la page actuelle
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get()
            }

        # Suivre le lien "Next"
        next_page = response.css('li.next a::attr(href)').get()
        if next_page:
            # Méthode 1 : URL relative
            yield response.follow(next_page, self.parse)

            # Méthode 2 : URL absolue
            # next_url = response.urljoin(next_page)
            # yield scrapy.Request(next_url, callback=self.parse)
```

Scrapy gère automatiquement :

- Dédoublonnage des URLs
- Profondeur de crawl
- File d'attente optimisée

Items - Définition des données

Fichier items.py

```
import scrapy

class QuoteItem(scrapy.Item):
    text = scrapy.Field()
    author = scrapy.Field()
    tags = scrapy.Field()
    author_url = scrapy.Field()

class BookItem(scrapy.Item):
    title = scrapy.Field()
    price = scrapy.Field()
    rating = scrapy.Field()
    availability = scrapy.Field()
    image_url = scrapy.Field()
```

Utilisation dans le spider

```
from myproject.items import QuoteItem

def parse(self, response):
    for quote in response.css('div.quote'):
        item = QuoteItem()
        item['text'] = quote.css('span.text::text').get()
        item['author'] = quote.css('small.author::text').get()
        item['tags'] = quote.css('a.tag::text').getall()
        yield item
```

ItemLoaders

Plus propre pour nettoyer les données

```
from scrapy.loader import ItemLoader
from scrapy.loader.processors import TakeFirst, MapCompose, Join
from myproject.items import QuoteItem

class QuoteLoader(ItemLoader):
    default_output_processor = TakeFirst()

    # Processors spécifiques
    tags_out = Join(', ')
    text_in = MapCompose(str.strip, str.replace, '\n', ' ')

# Dans le spider
def parse(self, response):
    for quote in response.css('div.quote'):
        loader = ItemLoader(item=QuoteItem(), selector=quote)

        loader.add_css('text', 'span.text::text')
        loader.add_css('author', 'small.author::text')
        loader.add_css('tags', 'a.tag::text')

        yield loader.load_item()
```

Avantages

- Code plus propre
- Transformations réutilisables
- Validation automatique

Pipelines - Traitement des données

Fichier pipelines.py

```python
import json
from itemadapter import ItemAdapter

class JsonWriterPipeline:
    """Sauvegarde en JSON"""

    def open_spider(self, spider):
        self.file = open('quotes.json', 'w', encoding='utf-8')
        self.items = []

    def close_spider(self, spider):
        json.dump(self.items, self.file, indent=2, ensure_ascii=False)
        self.file.close()

    def process_item(self, item, spider):
        self.items.append(ItemAdapter(item).asdict())
        return item

class PriceConversionPipeline:
    """Convertit les prix en float"""

    def process_item(self, item, spider):
        adapter = ItemAdapter(item)

        if adapter.get('price'):
            # Extraire le nombre du prix (ex: "£51.77" -> 51.77)
            price_text = adapter['price']
            adapter['price'] = float(''.join(filter(str.isdigit or '.' in price_text)))

        return item

class DuplicatesPipeline:
    """Supprime les doublons"""

    def __init__(self):
        self.ids_seen = set()

    def process_item(self, item, spider):
        adapter = ItemAdapter(item)
        if adapter['text'] in self.ids_seen:
            raise DropItem(f"Duplicate: {item}")
        else:
            self.ids_seen.add(adapter['text'])
            return item
```

Activer les Pipelines

Fichier settings.py

```
ITEM_PIPELINES = {
    'myproject.pipelines.DuplicatesPipeline': 100,
    'myproject.pipelines.PriceConversionPipeline': 200,
    'myproject.pipelines.JsonWriterPipeline': 300,
}
```

Priorité : Nombre plus petit = exécuté en premier (0-1000)

Pipeline avec condition

```
class ConditionalPipeline:
    def process_item(self, item, spider):
        # Appliquer seulement pour certain spider
        if spider.name == 'books':
            # Traitement
            pass
        return item
```

Export des données

Formats supportés

- JSON
- JSON Lines
- CSV
- XML
- Pickle

Via ligne de commande

```
# JSON
scrapy crawl quotes -O quotes.json
# CSV
scrapy crawl quotes -O quotes.csv
# JSON Lines (un item par ligne)
scrapy crawl quotes -O quotes.jl
# Avec encoding
scrapy crawl quotes -O quotes.json -t json --set FEED_EXPORT_ENCODING=utf-8
```

Via settings.py

```
FEEDS = {
    'data/%(name)s_%(time)s.json': {
        'format': 'json',
        'encoding': 'utf8',
        'store_empty': False,
        'indent': 2,
    },
}
```

Settings importants

```
# settings.py

# Identification
USER_AGENT = 'MyBot/1.0 (+http://example.com/bot)'
BOT_NAME = 'mybot'

# Politesse
CONCURRENT_REQUESTS = 16  # Requêtes simultanées
DOWNLOAD_DELAY = 2        # Secondes entre requêtes
AUTOTHROTTLE_ENABLED = True  # Ajustement automatique
AUTOTHROTTLE_START_DELAY = 1
AUTOTHROTTLE_MAX_DELAY = 10

# Respect de robots.txt
ROBOTSTXT_OBEY = True

# Retry
RETRY_ENABLED = True
RETRY_TIMES = 3
RETRY_HTTP_CODES = [500, 502, 503, 504, 408, 429]

# Logging
LOG_LEVEL = 'INFO'  # DEBUG, INFO, WARNING, ERROR
LOG_FILE = 'scrapy.log'

# Cookies
COOKIES_ENABLED = True

# Profondeur de crawl
DEPTH_LIMIT = 3
```

Middleware personnalisé

Rotation de User-Agents

```
# middlewares.py
import random

class RandomUserAgentMiddleware:
    def __init__(self):
        self.user_agents = [
            'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36...',
            'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36...',
            'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36...'
        ]

    def process_request(self, request, spider):
        request.headers['User-Agent'] = random.choice(self.user_agents)
```

Activer dans settings.py

```
DOWNLOADER_MIDDLEWARES = {
    'myproject.middlewares.RandomUserAgentMiddleware': 400,
}
```

Autres middlewares utiles

- Rotation de proxies
- Gestion de cookies
- Retry personnalisé
- Logging avancé

Scrapy Shell - Debug

Shell interactif pour tester sélecteurs

```
scrapy shell "http://quotes.toscrape.com"
```

Dans le shell

```
# response est automatiquement disponible
>>> response.url
>>> response.status

# Tester sélecteurs CSS
>>> response.css('div.quote').getall()
>>> response.css('span.text::text').get()

# Tester XPath
>>> response.xpath('//div[@class="quote"]').getall()

# Suivre un lien
>>> fetch('http://quotes.toscrape.com/page/2/')

# Voir HTML
>>> view(response)  # Ouvre dans le navigateur
```

Très utile pour :

- Développer des sélecteurs
- Débugger
- Explorer la structure

Bonnes pratiques Scraping

1. Légal et éthique

- Vérifier robots.txt
- Lire les CGU
- Respecter le rate limiting
- S'identifier (User-Agent)

2. Technique

- Gestion d'erreurs robuste
- Logging détaillé
- Tests unitaires
- Code modulaire

3. Performance

- Async avec Scrapy
- Cache HTTP
- Sélecteurs CSS optimisés
- Éviter les regex complexes

4. Maintenance

- Monitorer les changements
- Versionner le code
- Documenter les sélecteurs
- Tests de régression

Gestion des sites dynamiques (JavaScript)

Problème : Certains sites chargent le contenu avec JavaScript

Solutions

1. Scrapy + Selenium

```
from scrapy_selenium import SeleniumRequest

def start_requests(self):
    yield SeleniumRequest(
        url='https://example.com',
        callback=self.parse,
        wait_time=3
    )
```

2. Scrapy-Splash (service externe)

```
yield SplashRequest(
    url='https://example.com',
    callback=self.parse,
    endpoint='render.html'
)
```

3. Playwright (moderne, rapide)

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch()
    page = browser.new_page()
    page.goto('https://example.com')
    html = page.content()
```

Contournement de protections

⚠️ **Avertissement** : À utiliser de manière responsable et légale uniquement

Protections courantes

1. **Rate limiting** : Limitez vos requêtes, utilisez des proxies
2. **Captcha** : Services de résolution (coûteux), éviter si possible
3. **Cloudflare** : Complexe, nécessite souvent un navigateur headless
4. **User-Agent blocking** : Rotation de User-Agents
5. **IP blocking** : Rotation de proxies, VPN

Alternatives recommandées

- Contacter le site pour un accès API
- Respecter les limites et être transparent
- Utiliser des services d'agrégation de données

Règle d'or : Si un site ne veut pas être scrapé, respectez-le !

Monitoring et alertes

Surveiller vos scrapers

1. Logs structurés

```
import logging
logger = logging.getLogger(__name__)
logger.info(f"Scraped {count} items")
logger.error(f"Failed: {url}")
```

2. Métriques

- Nombre d'items scrapés
- Taux d'erreur
- Temps d'exécution
- URLs échouées

3. Alertes

- Email si taux d'erreur > 10%
- Slack notification si scraper échoue
- Monitoring avec Sentry, Datadog...

4. Tests de régression

- Vérifier que les sélecteurs fonctionnent toujours
- Comparer le nombre d'items scrapés
- Valider la structure des données

Déploiement en production

Options de déploiement

1. Scrapyd (serveur Scrapy)

```
pip install scrapyd
scrapyd
```

2. Cronjob

```
0 2 * * * cd /path/to/project && scrapy crawl myspider
```

3. Cloud (AWS, GCP, Azure)

- Lambda functions (petits scrapers)
- EC2 / Compute Engine (gros scrapers)
- Container (Docker + Kubernetes)

4. Scrapy Cloud (Zyte - payant)

- Hébergement spécialisé
- Monitoring intégré
- Proxies inclus

Bonnes pratiques

- Variables d'environnement pour config
- Logs centralisés
- Backup des données
- Alertes automatiques

Ressources et aller plus loin

Documentation

- Scrapy : https://docs.scrapy.org/
- BeautifulSoup : https://www.crummy.com/software/BeautifulSoup/
- Requests : https://requests.readthedocs.io/

Sites d'entraînement

- http://quotes.toscrape.com
- http://books.toscrape.com
- https://scrapethissite.com

Livres

- "Web Scraping with Python" - Ryan Mitchell
- "Python Web Scraping Cookbook" - Michael Heydt

Communauté

- Stack Overflow (tag: web-scraping)
- Reddit : r/webscraping
- Discord Scrapy

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Merci !

Questions ?

Annexe - Cheat Sheet BeautifulSoup

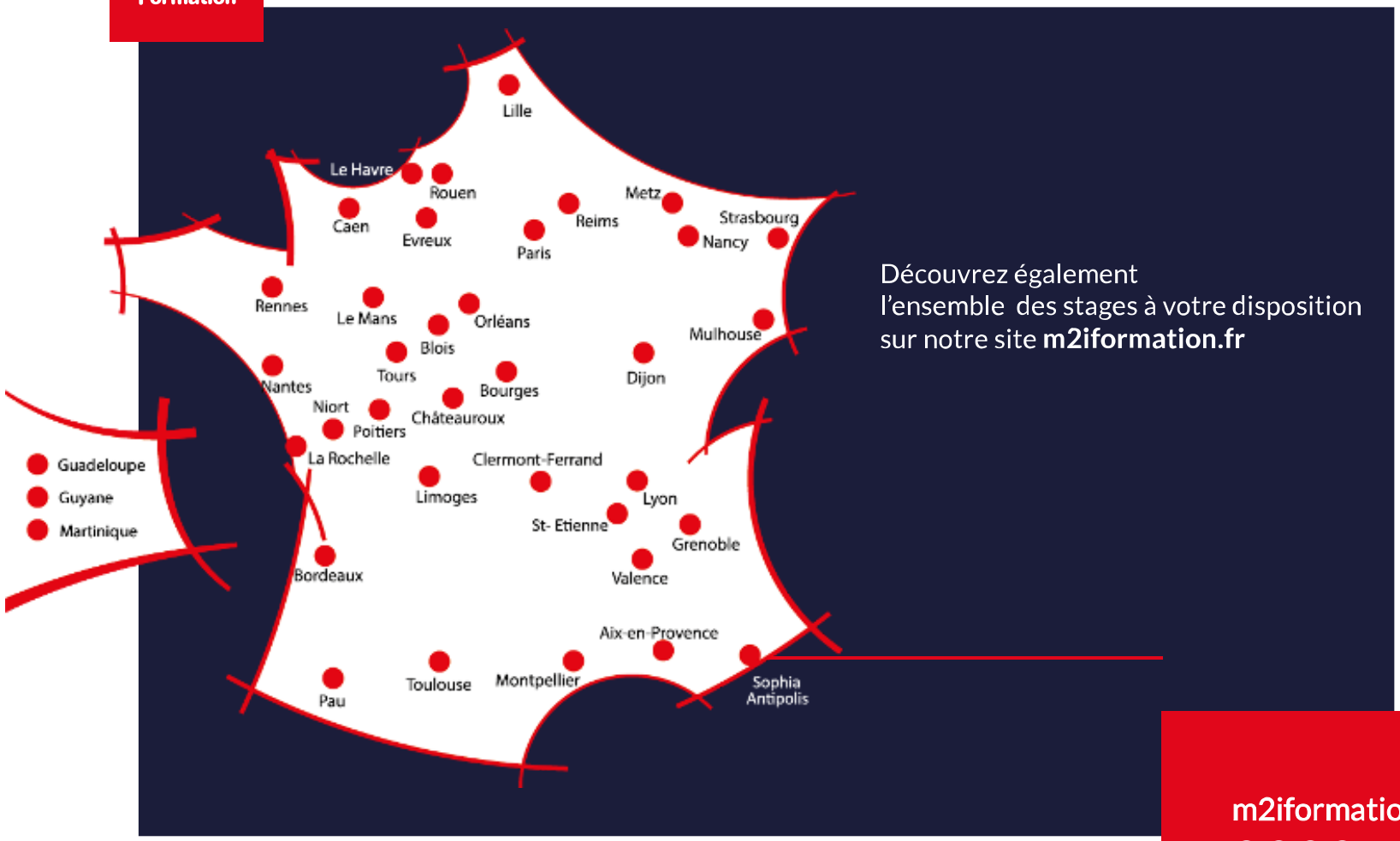
```
from bs4 import BeautifulSoup

# Parsing
soup = BeautifulSoup(html, 'lxml')

# Recherche
soup.find('div')                    # Premier div
soup.find('div', class_='content')  # Avec classe
soup.find(id='header')              # Par ID
soup.find_all('p')                  # Tous les p
soup.find_all(['p', 'div'])         # Plusieurs balises

# Sélecteurs CSS
soup.select('.class')               # Classe
soup.select('#id')                  # ID
soup.select('div p')                # p dans div
soup.select('div > p')              # p enfant direct

# Extraction
element.text                        # Texte
element.get_text()                  # Texte avec options
element['href']                     # Attribut
element.get('href')                 # Attribut (safe)
element.attrs                       # Tous les attributs

# Navigation
element.parent                      # Parent
element.children                    # Enfants directs
element.descendants                 # Tous descendants
element.next_sibling                # Frère suivant
element.previous_sibling            # Frère précédent
```

Annexe - Cheat Sheet Scrapy

```
# Créer projet
scrapy startproject myproject
scrapy genspider myspider example.com

# Lancer spider
scrapy crawl myspider
scrapy crawl myspider -O output.json

# Selectors
response.css('div.quote::text').get()      # Premier
response.css('div.quote::text').getall()   # Tous
response.css('a::attr(href)').get()        # Attribut
response.xpath('//div[@class="quote"]')    # XPath

# Spider basique
class MySpider(scrapy.Spider):
    name = "myspider"
    start_urls = ['http://example.com']

    def parse(self, response):
        yield {'data': response.css('...').get()}

        # Pagination
        next_page = response.css('a.next::attr(href)').get()
        if next_page:
            yield response.follow(next_page, self.parse)

# Items
class MyItem(scrapy.Item):
    field = scrapy.Field()

# Pipeline
class MyPipeline:
    def process_item(self, item, spider):
        # Traiter item
        return item

# Shell
scrapy shell "http://example.com"
>>> response.css('...').getall()
```

Annexe - Regex utiles

```python
import re

# Prix
re.findall(r'[\d.]+', '£51.77')  # ['51.77']

# Email
re.findall(r'[\w\.-]+@[\w\.-]+', text)

# URL
re.findall(r'https?://[^\s]+', text)

# Téléphone
re.findall(r'\d{2}[.\s-]?\d{2}[.\s-]?\d{2}[.\s-]?\d{2}[.\s-]?\d{2}', text)

# Supprimer HTML tags
re.sub(r'<[^>]+>', '', html)

# Nettoyer espaces
re.sub(r'\s+', ' ', text).strip()

# Extraction avec groupes
match = re.search(r'(\d+)/5', '4/5 étoiles')
if match:
    rating = match.group(1)  # '4'
```



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr