# Web scraping

Kossi

07/01/2020

## Introduction au Web scraping

Vous souvenez-vous de l'API de la banque mondiale que nous avions présenté au premier chapitre ? Dites-vous bien que nous n'avons pas encore fini avec elle. Dans cette partie nous allons nous attéler à convertir les données du format **xml/html** au format **json** qui est plus lisible et plus exploitable en *python*. Pour ce faire, nous allons utiliser principalement les modules python requests et BeautifulSoup. Nous avons déjà rencontré le premier module au chapitre 1. Quant au deuxième, il permet d'analyser ou, comme on préfère dire souvent, **parser** le code HTML des pages Web. Veuillez absolument revoir les notions introduites dans les deux premier chapitres car nous allons beaucoup les réutiliser dans celui-ci.

```
import requests
# Nous envoyons une requête `GET` à l'URL 'http://api.worldbank.org/v2/country/br'
reponse = requests.get("http://api.worldbank.org/v2/country/br")
code_etat = reponse.status_code
print("code  d'état : {}".format(code_etat))
```

```
## code  d'état : 200
```

```
contenu = reponse.text # Le contenu est au format xml
print("Contenu au format XML :\n", contenu)
```

```
## Contenu au format XML :
##  <?xml version="1.0" encoding="utf-8"?>
## <wb:countries page="1" pages="1" per_page="50" total="1" xmlns:wb="http://www.worldbank.org">
##   <wb:country id="BRA">
##     <wb:iso2Code>BR</wb:iso2Code>
##     <wb:name>Brazil</wb:name>
##     <wb:region id="LCN" iso2code="ZJ">Latin America &amp; Caribbean </wb:region>
##     <wb:adminregion id="LAC" iso2code="XJ">Latin America &amp; Caribbean (excluding high income)</wb
##     <wb:incomeLevel id="UMC" iso2code="XT">Upper middle income</wb:incomeLevel>
##     <wb:lendingType id="IBD" iso2code="XF">IBRD</wb:lendingType>
##     <wb:capitalCity>Brasilia</wb:capitalCity>
##     <wb:longitude>-47.9292</wb:longitude>
##     <wb:latitude>-15.7801</wb:latitude>
##   </wb:country>
## </wb:countries>
```

Si vous avez déjà oublié à quoi ressemble un format **xml**, le re-voilà ! Notre objectif sera, comme je le disais tantôt, de passer de ce format u format **json** qui s'affiche ci-dessous:

```
reponse = requests.get("http://api.worldbank.org/v2/country/br?format=json")
contenu = reponse.json()
print("Contenu au format JSON :\n", contenu)
```

```
## Contenu au format JSON :
```

```
##  [{'page': 1, 'pages': 1, 'per_page': '50', 'total': 1}, [{'id': 'BRA', 'iso2Code': 'BR', 'name': 'B
```

Comment allons-nous procéder ? C'est évidemment la question légitime que vous vous posez. Déjà si vous regardez les deux données, elles semblent contenir les mêmes infromations à savoir le nom du pays, son niveau de revenu, sa capitale ... Une première sera de passer du text brut contenu dans la reponse HTML, à un objet *python* beaucoup plus maniable.

```
from bs4 import BeautifulSoup

# Nous envoyons une requête `GET` à l'URL 'http://api.worldbank.org/v2/country/br'
reponse = requests.get("http://api.worldbank.org/v2/country/br")
contenu = reponse.text # Le contenu est au format xml
print("Contenu au format XML :\n", contenu)
```

```
## Contenu au format XML :
##  <?xml version="1.0" encoding="utf-8"?>
## <wb:countries page="1" pages="1" per_page="50" total="1" xmlns:wb="http://www.worldbank.org">
##   <wb:country id="BRA">
##     <wb:iso2Code>BR</wb:iso2Code>
##     <wb:name>Brazil</wb:name>
##     <wb:region id="LCN" iso2code="ZJ">Latin America &amp; Caribbean </wb:region>
##     <wb:adminregion id="LAC" iso2code="XJ">Latin America &amp; Caribbean (excluding high income)</wb
##     <wb:incomeLevel id="UMC" iso2code="XT">Upper middle income</wb:incomeLevel>
##     <wb:lendingType id="IBD" iso2code="XF">IBRD</wb:lendingType>
##     <wb:capitalCity>Brasilia</wb:capitalCity>
##     <wb:longitude>-47.9292</wb:longitude>
##     <wb:latitude>-15.7801</wb:latitude>
##   </wb:country>
## </wb:countries>
```

```
soup = BeautifulSoup(contenu, 'lxml-xml')
country = soup.country
print("Le  nom du pays est :", country.name)
```

```
## Le  nom du pays est : country
```

```
print("Son code ISO à 2 lettres est :", country.iso2Code)
```

```
## Son code ISO à 2 lettres est : <wb:iso2Code>BR</wb:iso2Code>
```

```
print("Sa capitale est :", country.capitalCity.text)
```

```
## Sa capitale est : Brasilia
```

```
print("La capitale a pour coordonnées géographiques:", (country.longitude, country.find("latitude")) )
```

```
## La capitale a pour coordonnées géographiques: (<wb:longitude>-47.9292</wb:longitude>, <wb:latitude>-
```

Le constructeur de la classe **BeautifulSoup** prend en entrée un document *HTML/XML* au format texte/binaire et renvoie un objet `bs4.BeautifulSoup` qui est un *arbre* modélisant notre document. On peut extraire un **tag** quelconque de cet arbre en faisanat **tag__parent.tag__enfant**. Cette façon de faire est un "racourci" et est plus convenable que son équivalente mais plus générale **tag__parent**.*find*("**tag__enfant**"). Une dernière possibilité consiste à faire **tag__parent.tag__enfant.text**. Les deux premiers syntaxes nous simplifient la vie en nous évitant d'ajouter le **.text**.

```
Note : Il est aussi possible d'accéder aux tags enfants en faisant tag_parent.find("{tag_enfant}"). Cet
```

Mieux, une fois le tag extrait, nous pouvons récupérer non seulement son texte mais aussi ses attributs (vous vous rappelez des attributs HTML ?). Cela se fait en faisant **tag.attrs**["**nom__attribut**"]. Veuillez regarder le bout de code ci-dessous pour mieux comprendre.

```
region = country.region
region_id = region.attrs["id"]
region_iso2code = region.attrs["iso2code"]
print("Ce pays est dans la zone '{}'. Cette zone a pour ID '{}' et pour code ISO APLHA 2 '{}'.".format(
```

```
## Ce pays est dans la zone 'Latin America & Caribbean '. Cette zone a pour ID 'LCN' et pour code ISO A
```

```
Exercice1 : En vous inspirant du message ci-dessus, afficher un message similaire pour l'élément 'lendi
```

```
Exercice2 : En se basant sur tout ce qui précède, écrire une fonction qui prend en entrée le code XML d
```

## Et si l'arborescence était un peu plus complexe ?

Un des avantages de la méthode vue précédemment est que l'on peut accéder aux champs de l'élément XML/HTML comme on le ferait pour n'importe quel objet python. Mais cette approche devient rapidemnt long et inutilement complexe dès que l'élément d'intérêt est un peu caché dans la structure arborescente de la page; et c'est souvent le cas sur Internet où les pages HTML dépassent facilement des dizaines de pages en code source.

Considérons ce bout de code HTML. Comment allez-vous extraire l'attribut **title** du conteneur **div** le plus profond ?

```
html = """
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Un exemple de document profond</title>
    <!-- On peut avoir d'autres méta-données ici -->
  </head>
  <body>
    <!-- Ici, on placera tout le contenu à destination de l'utilisateur -->
    <div class="conteneur exterieur" id="racine" title="Je suis le premier div." >
        <span class="debut" data="Voici un element span">Niveau 1</span>
        <div class="conteneur interieur" title="Mon predecesseur est un div avec id.">
            <span>Niveau 2</span>
            <div class="conteneur interieur milieu" title="Je suis un div sans attribut id. Je suis au
                <p class="paragraphe milieu" type="paragraphe-milieu">Ecrire un paragraph ici.</p>
                <span class="milieu">Niveau 3</span>
                <div class="conteneur interieur">
                    <span>Niveau 4</span>
                    <div class="conteneur interieur fin" id="feuille" title="Extraire mon contenu.">
                        <p class="paragraphe fin" type="paragraphe-fin">Paragraphe de fin ici.</p>
                        <span class="fin">Niveau 5</span>
                    </div>
                </div>
            </div>
        </div>
    </div>
  </body>
</html>
"""
```

La méthode vue précédemment nous conduit à une solution du type:

```
soup =  BeautifulSoup(html)
deepest_div = soup.div.div.div.div.div
deepest_div_title = deepest_div.attrs["title"]
print("Le champ 'title' du plus profond élémetn 'div' est :", deepest_div_title)
```

```
## Le champ 'title' du plus profond élémetn 'div' est : Extraire mon contenu.
```

Vous convenez avec moi que cette solution qui utilise des éléments *divs* en cascade n'est sûrement pas la meilleure du marché. Effectivement il y a beaucoup plus simple. Il s'agit en fait de profiter du fait que l'élément div qui nous intéresse possède un champ **id**. La valeur d'un champ **id** est souvent supposée unique dans un document HTML et permet de retrouver très rapidement l'élément qui la porte. Dans notre cas par exemple, ce champ ID nous permettrait d'écrire un code qui ressemblerait à ça:

```
soup =  BeautifulSoup(html)
deepest_div = soup.find("div", id="feuille") # Beaucoup plus joli ;(
deepest_div_title = deepest_div.attrs["title"]
print("Le champ 'title' du plus profond élémetn 'div' est :", deepest_div_title)
```

```
## Le champ 'title' du plus profond élémetn 'div' est : Extraire mon contenu.
```

Cette façon d'utiliser la fonction **find** n'est pas spécifique au champ **id**. On peut le faire pratiquement avec tous les champs. Par exemple avec le champ **type** on aurait:

```
soup = BeautifulSoup(html)
middle_p = soup.find("p", type="paragraphe-milieu")
middle_p_class = middle_p.attrs["class"]
print("La classe du paragraphe du milieu est:", middle_p_class)
```

```
## La classe du paragraphe du milieu est: ['paragraphe', 'milieu']
```

### Sélectionner un tag possédant certains attributs

Pour sélectionner un tag possédant un certain attribut, on utilise la fonction **find** ou **find_all** combiné soit avec le nom des attributs soit avec l'argument **attrs**.

```
# Sélectionner un élément `div` avec une classe 'interieur" mais sans attribut `title`
# Remarquer l'usage de find_all qui renvoie toute la liste des tags qui vérifient les critères spécifié
div_interieur_sans_titre = soup.find_all("div", attrs = {"class": "interieur", "title": False})
print("Elément(s) div avec une classe 'interieur mais sans attribut `title` est :\n", div_interieur_sans
```

```
## Elément(s) div avec une classe 'interieur mais sans attribut `title` est :
##  [<div class="conteneur interieur">
## <span>Niveau 4</span>
## <div class="conteneur interieur fin" id="feuille" title="Extraire mon contenu.">
## <p class="paragraphe fin" type="paragraphe-fin">Paragraphe de fin ici.</p>
## <span class="fin">Niveau 5</span>
## </div>
## </div>]
```

```
print("Un enfant `span` de ce(s) `div`  est :\n", div_interieur_sans_titre[0].span)
```

```
## Un enfant `span` de ce(s) `div`  est :
##  <span>Niveau 4</span>
```

### Et si les attributs ne sont que partiellement connus ?

```python
# Sélectionner un élément `div` l'attribut `title` contient la chaine "contenu"
div_interieur_sans_titre = soup.find("div", title = lambda x: x and "contenu" in x)
print("Elément div avec un titre contenant la chaîne 'contenu' :\n", div_interieur_sans_titre)
```

```
## Elément div avec un titre contenant la chaîne 'contenu' :
##  <div class="conteneur interieur fin" id="feuille" title="Extraire mon contenu.">
## <p class="paragraphe fin" type="paragraphe-fin">Paragraphe de fin ici.</p>
## <span class="fin">Niveau 5</span>
## </div>
```

## Une interface commune et générale : les sélecteurs CSS

Vous vous souvenez du langage **CSS** dont nous avons parlé au chapitre 1 ? Pour pouvoir mettre en forme les pages, le **CSS** met à disposition des developpeurs des sélecteurs dit **sélecteurs css**. Ces sélecteurs rendent l'analyse d'un document **XML/HTML** beaucoup plus compact. En *python* et avec **BeautifulSoup**, les sélecteurs CSS sont accessible via les fonctions **select_one** et **select**. Tout comme **find**, **select_one** renvoie le premier élément qui vérifie les critères spécifiés. De même, tout comme **find_all**, **select** renvoie tous les éléments solutions.

### Et si les attributs ne sont que partiellement connus ?

```python
# Sélectionner un élément p
p_element = soup.select_one("p")
print("Un élément p :", p_element)
```

```
## Un élément p : <p class="paragraphe milieu" type="paragraphe-milieu">Ecrire un paragraph ici.</p>
```

```python
div_with_title = soup.select_one("div[title]")
print("Un élément div avec un attribut `title` :\n", div_with_title)
```

```
## Un élément div avec un attribut `title` :
##  <div class="conteneur exterieur" id="racine" title="Je suis le premier div.">
## <span class="debut" data="Voici un element span">Niveau 1</span>
## <div class="conteneur interieur" title="Mon predecesseur est un div avec id.">
## <span>Niveau 2</span>
## <div class="conteneur interieur milieu" title="Je suis un div sans attribut id. Je suis au milieu.">
## <p class="paragraphe milieu" type="paragraphe-milieu">Ecrire un paragraph ici.</p>
## <span class="milieu">Niveau 3</span>
## <div class="conteneur interieur">
## <span>Niveau 4</span>
## <div class="conteneur interieur fin" id="feuille" title="Extraire mon contenu.">
## <p class="paragraphe fin" type="paragraphe-fin">Paragraphe de fin ici.</p>
## <span class="fin">Niveau 5</span>
## </div>
## </div>
## </div>
## </div>
## </div>
```

```python
div_with_title_containing_contenu = soup.select_one("div[title*='contenu']")
print("Un élément div avec un attribut `title` contenant la chaîne 'contenu':\n", div_with_title_contai
```

```
## Un élément div avec un attribut `title` contenant la chaîne 'contenu':
##  <div class="conteneur interieur fin" id="feuille" title="Extraire mon contenu.">
## <p class="paragraphe fin" type="paragraphe-fin">Paragraphe de fin ici.</p>
```

```
## <span class="fin">Niveau 5</span>
## </div>
```

```
interior_div_with_title_containing_suis = soup.select_one("div[title*='suis'][class*='interieur']")
print("Un élément div `interieur` avec un attribut `title` contenant la chaîne 'suis':\n", interior_div
```

```
## Un élément div `interieur` avec un attribut `title` contenant la chaîne 'suis':
##  <div class="conteneur interieur milieu" title="Je suis un div sans attribut id. Je suis au milieu."
## <p class="paragraphe milieu" type="paragraphe-milieu">Ecrire un paragraph ici.</p>
## <span class="milieu">Niveau 3</span>
## <div class="conteneur interieur">
## <span>Niveau 4</span>
## <div class="conteneur interieur fin" id="feuille" title="Extraire mon contenu.">
## <p class="paragraphe fin" type="paragraphe-fin">Paragraphe de fin ici.</p>
## <span class="fin">Niveau 5</span>
## </div>
## </div>
## </div>
```

Je vous recommande vivement de visiter cette page pour d'amples inforomations sur les **sélecteurs** CSS.

## Une interface encore plus générale : les XPATH

Les **Xpath** ont la même fonctionalité que les **sélecteurs CSS** mais sont bien plus généraux et permettent d'écrire des expressions beaucoup plus complexes. Ils sont par contre lents à processer. L'usage veut que l'on ne passe aux Xpaths que lorsqu'il est vraiment impossible d'écrire 'écrire en CSS. Vous pouvez visiter cette page pour apprendre davantage sur la syntaxe **Xpath**. Nous ne l'abordons pas ici par ce qu'il n'est pas supporté par BeautifulSoup. Mais avec les pages dynamiques et le module **selenium**, nous les rencontrerons sûrement.

## Résumé:

Dans cette troisième partie, nous avons aborder:

- Le Web scraping avec python et les modules **requests** et **BeautifulSoup**
- La syntaxe **BeautifulSoup** et son interface de navigation dans un document **XML/HTML**
- Les **sélecteurs CSS** comme outil générique de sélection d'éléments HTML