Scraping

Quelques autres méthodes du type `str`

Cleaning

Entrée [1]:

```
# Ma string s contient:
# - des espaces
# - des tabulations (\t)
# - des retours à la ligne (\n)
s = "\t hello\nworld  "
print(s)
```

```
         hello
world
```

Entrée [2]:

```
# Remplacer le retour à la ligne par un espace:
print(s.replace('\n', ' '))
```

```
         hello world
```

Entrée [3]:

```
# Enlever les caractères blancs au début et à la fin de la string:
print(s.strip())
```

```
hello
world
```

Entrée [4]:

```
cleaned = s.replace('\n', ' ').strip()
print(cleaned)
```

```
hello world
```

Trouver une str dans une autre

Entrée [5]:

```
cleaned.startswith('he')
```

Out[5]:

```
True
```

Entrée [6]:

```
cleaned.endswith('orld')
```

Out[6]:

```
True
```

Entrée [7]:

```
"wo" in cleaned
```

Out[7]:

```
True
```

Entrée [8]:

```
cleaned.index("wor")  # renvoie la position de "wor" dans "hello world"
```

Out[8]:

```
6
```

str.join

Permet de joindre une liste de strings.

Entrée [9]:

```
" ".join(['how', 'are', 'you'])
```

Out[9]:

```
'how are you'
```

Entrée [10]:

```
"---".join(['how', 'are', 'you'])
```

Out[10]:

```
'how---are---you'
```

Dans jupyterlab, pensez à utiliser `<Tab>` pour l'autocomplétion et pour découvrir quelles méthodes un type possède. `<Shift> + <Tab>` pour afficher la documentation d'une fonction.

Je vous invite aussi à survoler les fonctions présentes de base en python:

- https://docs.python.org/3/library/functions.html (https://docs.python.org/3/library/functions.html)

Ainsi que les modules présents de base (stdlib):

- https://docs.python.org/3/library/ (https://docs.python.org/3/library/)

Et pour tout le reste il y a pypi:

- http://pypi.org (http://pypi.org)

Entrée [11]:

```
# "pip install requests" pour pouvoir l'utiliser
import requests

response = requests.get('https://fr.wikipedia.org/plop')
print(response)
```

```
<Response [404]>
```

Entrée [12]:

```
print(response.status_code)
```

```
404
```

Entrée [13]:

```
page = "https://www.beerwulf.com/fr-fr/p/bieres/brasserie-de-sutter-brin-de-folie.3
content = requests.get(page).text
print(content[:100])
```

```
<!doctype html>
<html class="no-js" lang="fr-FR"
      data-original-lang="fr-FR"
      data-re
```

Pour voir le code source HTML d'une page dans le navigateur:

- Ctrl-U
- ou F12 > Elements

Entrée [14]:

```
before_price = '<span class="price">'
idx = content.index(before_price)
price = content[idx+len(before_price):idx+100]
price = price.split('<')[0]
```

Entrée [15]:

```
price
```

Out[15]:

```
'€ 2,29'
```

Entrée [16]:

```
print(float(price[2:].replace(',', '.')))
```

```
2.29
```

Moralité

C'est un peu le bordel d'extraire des infos du document html en manipulant le document comme une bête str.

Here comes beautifulsoup:

Entrée [17]:

```
from bs4 import BeautifulSoup
```

Je vous invite à tester le HTML suivant sur https://html.house (https://html.house).

(Cours HTML/CSS sur openclassrooms (https://openclassrooms.com/fr/courses/1603881-apprenez-a-creer-votre-site-web-avec-html5-et-css3))

Entrée [18]:

```
html = """
<html>
    <head>
        <style>
        h1 { font-size: 50px; }
        body { font-family: Verdana; }
        li { color: red; }
        ul ul li { color: green; }
        .highlighted { font-weight: bold; }
        .italic { font-style: italic; }
        .highlighted.italic { }
        </style>
    </head>
    <body>
        <h1>Mon titre</h1>
        <p class="highlighted">
            Some text with a<br>
            <a href="https://google.com">link to google</a>
            <img src="https://picsum.photos/200/300">
        </p>
        <p>Some list:</p>
        <ul>
            <li>some item</li>
            <li class="highlighted italic">some item</li>
            <li class="italic">some item</li>
            <ul>
                <li>some other item 1</li>
                <li>some other item 2</li>
            </ul>
            <li>some item</li>
        </ul>
    </body>
</html>
"""
```

Entrée [19]:

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html)
```

Entrée [20]:

```
titre = soup.find('h1')
```

Entrée [21]:

```
titre
```

Out[21]:

```
<h1>Mon titre</h1>
```

Entrée [22]:

```
type(titre)
```

Out[22]:

```
bs4.element.Tag
```

Entrée [23]:

```
titre.text
```

Out[23]:

```
'Mon titre'
```

Entrée [24]:

```
titre.name
```

Out[24]:

```
'h1'
```

Entrée [25]:

```
link = soup.find('a')
link.attrs
```

Out[25]:

```
{'href': 'https://google.com'}
```

Entrée [26]:

```
paragraph = soup.find('p')
```

Entrée [27]:

```
paragraph
```

Out[27]:

```
<p class="highlighted">
            Some text with a<br/>
<a href="https://google.com">link to google</a>
<img src="https://picsum.photos/200/300"/>
</p>
```

Entrée [28]:

```
paragraph.find('img')
```

Out[28]:

```
<img src="https://picsum.photos/200/300"/>
```

Entrée [29]:

```
soup.find_all('li', class_="italic")
```

Out[29]:

```
[<li class="highlighted italic">some item</li>,
 <li class="italic">some item</li>]
```

Entrée [30]:

```
# La même chose qu'au dessus, mais à l'aide d'un sélecteur css:
soup.select('li.italic')
```

Out[30]:

```
[<li class="highlighted italic">some item</li>,
 <li class="italic">some item</li>]
```

Entrée [31]:

```
# Récupérer les li de 2e niveau (qui sont dans un ul lui-même dans un ul)
soup.find('ul').find('ul').find_all('li')
```

Out[31]:

```
[<li>some other item 1</li>, <li>some other item 2</li>]
```

Entrée [32]:

```
# Même chose avec un sélecteur css:
soup.select('ul ul li')
```

Out[32]:

```
[<li>some other item 1</li>, <li>some other item 2</li>]
```

Entrée [33]:

```
li = soup.select('ul ul li')[0]
li
```

Out[33]:

```
<li>some other item 1</li>
```

Entrée [34]:

```
li.find_next_sibling()
```

Out[34]:

```
<li>some other item 2</li>
```

Entrée [35]:

```
li.parent
```

Out[35]:

```
<ul>
<li>some other item 1</li>
<li>some other item 2</li>
</ul>
```

Entrée [36]:

```
li.parent.contents
```

Out[36]:

```
['\n', <li>some other item 1</li>, '\n', <li>some other item 2</li>,
'\n']
```

Entrée [37]:

```
li.parent.find_all()  # même chose que .contents mais renvoie uniquement les Tag
```

Out[37]:

```
[<li>some other item 1</li>, <li>some other item 2</li>]
```

Cas concret: beerwulf.com

Entrée [38]:

```
from bs4 import BeautifulSoup

def get_soup_from_url(url):
    page = requests.get(url)
    return BeautifulSoup(page.text)

def extract_beer_infos(url):
    soup = get_soup_from_url(url)

    # Extract price:
    price = soup.select('span.price')[0].text
    price = float(price[2:].replace(',', '.'))  # "€ 2,29" => 2.29

    # Extract volume:
    volume = soup.find('dt', text='Contenu').find_next_sibling()
    volume = int(volume.text[:-2])  # "33cl" => 33

    # Extract evaluation:
    note = soup.find('div', class_='stars')
    note = int(note.attrs['data-percent'])

    # Extract EBC:
    ebc = soup.find('div', class_='ebc')
    children = ebc.find_all('div')
    active_tag = ebc.find('div', class_='active')
    position = children.index(active_tag)
    ebc_pct = position / len(children) * 100

    infos = {
        'price': price,
        'volume': volume,
        'note': note,
        'ebc': ebc_pct,
    }
    return infos
```

Entrée [39]:

```
extract_beer_infos("https://www.beerwulf.com/fr-fr/p/bieres/brugse-zot-blond2")
```

Out[39]:

```
{'price': 2.29, 'volume': 33, 'note': 70, 'ebc': 7.6923076923076925}
```

Inconvénients du scraping:

1) le contenu généré dynamiquement en javascript n'est pas présent initialement sur la page, ce qui fait qu'on peut avoir un contenu différent entre ce qu'on trouve dans l'inspecteur (sur chrome: F12 > Elements)

Entrée [40]:

```
soup = get_soup_from_url("https://www.beerwulf.com/fr-fr/c/bieres/style/Blonde")
```

Entrée [41]:

```
# Ce find ne renvoie rien car le products-container est ajouté via Javascript
soup.find('div', class_="products-container")
```

Un exemple minimal de page où le code source de la page ne contient pas les données initialement:
https://kim.fspot.org/cours/page4.html (https://kim.fspot.org/cours/page4.html)

Entrée [42]:

```
infos = [
    extract_beer_infos('https://www.beerwulf.com/fr-fr/p/bieres/brouwerij-t-verzet-
    extract_beer_infos('https://www.beerwulf.com/fr-fr/p/bieres/bruxellensis2')
]
```

Entrée [43]:

```
infos
```

Out[43]:

```
[{'price': 1.99, 'volume': 33, 'note': 70, 'ebc': 15.384615384615385},
 {'price': 3.49, 'volume': 33, 'note': 70, 'ebc': 15.384615384615385}]
```

Entrée [44]:

```
import pandas as pd
df = pd.DataFrame(infos)
```

Entrée [45]:

```
df
```

Out[45]:

	ebc	note	price	volume
0	15.384615	70	1.99	33
1	15.384615	70	3.49	33

Entrée [46]:

```
df[df.price < 2]
```

Out[46]:

	ebc	note	price	volume
0	15.384615	70	1.99	33

2) inconvénient du scraping: on est tributaire de l'architecture du HTML. Si les développeurs du site web changent le design, il y a de fortes chances que le programme beautifulsoup doive être réécrit.

Solution: **préférer une API**