DataScientest • com

Web scraping with Python

Request and BeautifulSoup

In order to achieve the final goal of the web scraping, it is first necessary to recover the source code from a web page. In a second step, it is important to understand the structure of the recovered code to be able to navigate and collect the data we care about.

The objectives of this notebook are as follows:

- Create an object `BeautifulSoup`
- Navigate in the tree of an HTML document using the tags.

The notebook is based on the example of the Amazon page of best books available here (https://www.amazon.com.au/gp/bestsellers/?ref_=nav_cs_bestsellers).

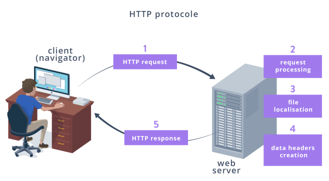
If the link does not work, it is possible to access the page by going to the site via your browser, or carry out the exercise from another page.

Introduction to Request

1. HTTP

The HTTP protocole (Hypertext Transfer Protocol) is a protocol used by the web to establish communication between customers and servers or, more simply, between the web browser of a person consulting a site and the machine hosting the site.
For the customer to access the site's resources, he uses the HTTP protocol. Depending on the type of request, it uses different methods asking the server to perform the requested action and display the requested resource.
Below is three examples of methods used by the protocol:

- `GET` : recover the content of a resource (using the URL)
- `PUT` : create or replace the content of a resource.
- `HEAD` : request only information on the resource, without asking for the resource itself.



This protocol is essential to access the content of an external resource. It is also identifiable in the URL link of web pages.

2. Request

The `Request` module is a Python module allowing easily to make the HTTP protocol.
As part of the Scraping Web, we must make a request to recover the source code of the page.
The `urllib` module of the `Request` module allows very easily to carry out this step using its `urlopen()` function. The content of the web page you want to scrap is then recoverable by the function by entering the link.

- **a)** Import the `urlopen` function of the `Request` module.
- **b)** Recover and store the HTML code of the web page in a variable named `page`.

In [17]:

```python
import urllib.request
from bs4 import BeautifulSoup

amazon_url = "https://www.amazon.com.au/gp/bestsellers/books/ref=zg_bs_nav_0"
page = urllib.request.urlopen(amazon_url)
print(page)
```

```
<http.client.HTTPResponse object at 0x7ff9a5054a30>
```

Hide solution

In [25]:

```python
import urllib.request
from bs4 import BeautifulSoup

amazon_url = "https://www.amazon.com.au/gp/bestsellers/books/ref=zg_bs_nav_0"
page = urllib.request.urlopen(amazon_url)
```

`BeautifulSoup` is a tool that requires dependencies. Indeed, before any use of this bookstore, it is necessary to call on `Request` and go through the previous content recovery stage.

BeautifulSoup

`BeautifulSoup` is a Python module allowing to extract HTML file data. It is compatible with the vast majority of browsers and allows you to automate web data collection.

Once the source code has been obtained, the second step to write a web-scraper is to convert this code into a `BeautifulSoup` object to be able to exploit it.
To decipher this source code there are several ways to create a Soup (the BeautifulSoup object).
`BeautifulSoup` provides different synthetic analyzers called parsers which will allow the page to be transformed into an usable document. According to the parsers chosen (html.parser, lxml, lxml-xml, xml, html5lib), the module created, from the same document, different trees depending on the language chosen. Note, that all the languages of the various analyzers are quite similar, with the exception of HTML Parsers and XML Parsers.

- `soup = BeautifulSoup(page, 'html.parser')`
- `soup = BeautifulSoup(page, 'xml.parser')`

The web scraping method with this tool is a so - called static method because it ignores JavaScript and only exploits the HTML and CSS code on the web page considered.

In [19]:
```python
### Insert your solution

from bs4 import BeautifulSoup as bs

soup = bs(page, 'html.parser')

print(soup)
```

```
<!DOCTYPE html>
<html class="a-no-js" data-19ax5a9jf="dingo" lang="en-au"><!-- sp:feature:head-start -->
<head><script>var aPageStart = (new Date()).getTime();</script><meta charset="utf-8"/>
<!-- sp:end-feature:head-start -->
<!-- sp:feature:csm:head-open-part1 -->
<!-- sp:end-feature:csm:head-open-part1 -->
<!-- sp:feature:cs-optimization -->
<meta content="on" http-equiv="x-dns-prefetch-control"/>
<link href="https://images-fe.ssl-images-amazon.com" rel="dns-prefetch"/>
<link href="https://m.media-amazon.com" rel="dns-prefetch"/>
<link href="https://completion.amazon.com" rel="dns-prefetch"/>
<!-- sp:end-feature:cs-optimization -->
<!-- sp:feature:csm:head-open-part2 -->
<!-- sp:end-feature:csm:head-open-part2 -->
<!-- sp:feature:aui-assets -->
<link href="https://m.media-amazon.com/images/I/11EIQ5IGqaL._RC|01ZTHTZObnL.css,410yLeQZHKL.css,31OSFXVtM5L.css,013z33uKh2L.cs
s,017DsKjNQJL.css,0131vqwP5UL.css,41EWOOlBJ9L.css,11TIuySqr6L.css,01ElnPiDxWL.css,11fJbvhE5HL.css,01Dm5eKVxwL.css,01IdKcBuAdL.c
ss,01y-XAlI+2L.css,21P6CS3L9LL.css,01oDR3IULNL.css,41Js1DVdbVL.css,01XPHJk60-L.css,01S0vRENeAL.css,21IbH+SoKSL.css,11MrAKjcAKL.
css,21fecG8pUzL.css,11a5wZbuKrL.css,01CFUgsA-YL.css,31pHA2U5D9L.css,11qour3ND0L.css,116t+WD27UL.css,11gKCCKQV+L.css,11061HxnEv
```

Hide solution

In [26]:
```python
from bs4 import BeautifulSoup as bs

soup = bs(page, "html.parser")
```

If you display the Soup object, you get the HTML tree on the web page. To improve the structure and to have better visibility of the tree, you can use the `prettify()` method.

- **e)** Display the parser and identify five tags.

In [23]:
```python
### Insert your solution

print(soup)
```

Show solution

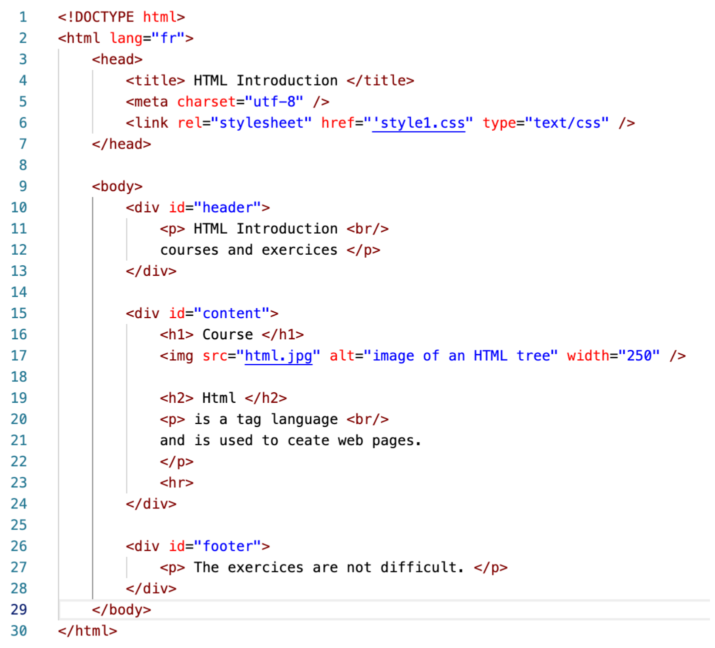
The parser of a web page is, in practice, quite difficult to read. Thus, to understand the structure of the code, we will base ourselves on an example of a less complex HTML document.

HTML tree navigation

Tree structure

An HTML page is organized as a tree structure of subcategories structured by tags also called TAG or HTML elements.

Example of an HTML tree



```html
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title> HTML Introduction </title>
        <meta charset="utf-8" />
        <link rel="stylesheet" href="'style1.css" type="text/css" />
    </head>

    <body>
        <div id="header">
            <p> HTML Introduction <br/>
            courses and exercices </p>
        </div>

        <div id="content">
            <h1> Course </h1>
            <img src="html.jpg" alt="image of an HTML tree" width="250" />

            <h2> Html </h2>
            <p> is a tag language <br/>
            and is used to ceate web pages.
            </p>
            <hr>
        </div>

        <div id="footer">
            <p> The exercices are not difficult. </p>
        </div>
    </body>
</html>
```

Tags

The pairs tags: open, contain text, and close. The character `/` points out that it is a closure tag.

Examples of html pairs

Tag	Role
`<a> </a>`	Hyperlink
`<h1> </h1>, <h2>...</h6>`	Section titles from level 1 to 6
`<div> </div>`	Division of content, it serves as a container for shaping in CSS.
`<p> </p>`	Paragraph
`<ul> </ul>, <ol> </ol>`	List with unnumbered and numbered bullets
`<li> </li>`	Bullet list items
`<table> </table>`	Table
`<tr> </tr>`	Table row
`<td> </td>`	Table cell
`<tbody> </tbody>`	Table body (groups one or more tags `<tr>`)
`<script>...</script>`	JavaScript code

Orphan tags are tags that are used to insert an element in a specific location (for example an image, a link). It is not necessary to delimit the beginning and the end of this order, it is simply written as this: `<tag />` .

Examples of HTML orphan tags

Tag	Role
`<img/>`	Image
`<br/>`	Return to line in a paragraph mainly
`<link/>`	Introduce a link to a CSS or JS file

Navigation

The objective being to recover the data from the tree, we seek to access the different tags. To do this, we can navigate in the tree thanks to the name of the tags.
For example to have the Amazon page title tag, you can use the following code:

```
soup.title
```

In [33]:

```python
### Insert your solution
print(soup.h1.text)
```

```
Best Sellers in Books
```

In [39]:

```python
soup
```

Out[39]: https://www.amazon.com.au/gp/bestsellers/books/ref=zg_bs_nav_0 (https://www.amazon.com.au/gp/bestsellers/books/ref=zg_bs_nav_0)

Hide solution

In [28]:
```
print(soup.h1.text)

soup.div.div.a # The tag is the 1st tag at the time of creating the course
```

```
Best Sellers in Books
```

Out[28]:
```
<a aria-label="Amazon.com.au" class="nav-logo-link nav-progressive-attribute" href="/ref=nav_logo" id="nav-logo-sprites">
<span class="nav-sprite nav-logo-base"></span>
<span class="nav-sprite nav-logo-ext nav-progressive-content" id="logo-ext"></span>
<span class="nav-logo-locale">.com.au</span>
</a>
```

> Any text within a tag is considered an element of the HTML document. To recover it, it is possible to use the texts `text` or `string` .

- **h)** Recover the title of the page.

In [32]:
```
### Insert your solution


soup.title.string
```

Out[32]:
```
'Amazon.com.au Best Sellers: The most popular items in Books'
```

Hide solution

In [29]:
```
soup.title.string
```

Out[29]:
```
'Amazon.com.au Best Sellers: The most popular items in Books'
```

> The HTML tree is built from a 'parent' and its 'children'. A tag that contains other tags is the parent of the latter. And on the other hand, the tags contained are the children of the parent element.
> A child is not necessarily a tag but any element that an element can contain.
>
> In the example of the tree above, the `string` "Introduction to HTML" is the child of `title` which has as parent `head` .
>
> To navigate in the tree of a parent to a child or to access the descendants of a parent (in other words to access all the children of a parents), it is possible to use the generators `parent` , `children` (or `contents`) and `descendants` . Except for `contents` , the information in these orders must be recovered from a list, otherwise it cannot be read.

- **i)** Access the parent from the HTML element of the page. What do you notice ?
- **j)** What is the number of children in the `BeautifulSoup` object ?

In [31]:
```
### Insert your solution

print(list(soup.html.parent))
print("The BeautifulSoup object has children")
print("The beautifulSoup object has ", len(list(soup.descendants)), "childrens")
```

```
['html', <html class="a-no-js" data-19ax5a9jf="dingo" lang="en-au"><!-- sp:feature:head-start -->
<head><script>var aPageStart = (new Date()).getTime();</script><meta charset="utf-8"/>
<!-- sp:end-feature:head-start -->
<!-- sp:feature:csm:head-open-part1 -->
<!-- sp:end-feature:csm:head-open-part1 -->
<!-- sp:feature:cs-optimization -->
<meta content="on" http-equiv="x-dns-prefetch-control"/>
<link href="https://images-fe.ssl-images-amazon.com" rel="dns-prefetch"/>
<link href="https://m.media-amazon.com" rel="dns-prefetch"/>
<link href="https://completion.amazon.com" rel="dns-prefetch"/>
<!-- sp:end-feature:cs-optimization -->
<!-- sp:feature:csm:head-open-part2 -->
<!-- sp:end-feature:csm:head-open-part2 -->
<!-- sp:feature:aui-assets -->
<link href="https://m.media-amazon.com/images/I/11EIQ5IGqaL._RC|01ZTHTZObnL.css,410yLeQZHKL.css,31OSFXVtM5L.css,013z33uKh2L.cs
s,017DsKjNQJL.css,0131vqwP5UL.css,41EWOOlBJ9L.css,11TIuySqr6L.css,01ElnPiDxWL.css,11fJbvhE5HL.css,01Dm5eKVxwL.css,01IdKcBuAdL.c
ss,01y-XAlI+2L.css,21P6CS3L9LL.css,01oDR3IULNL.css,41Js1DVdbVL.css,01XPHJk60-L.css,01S0vRENeAL.css,21IbH+SoKSL.css,11MrAKjcAKL.
css,21fecG8pUzL.css,11a5wZbuKrL.css,01CFUgsA-YL.css,31pHA2U5D9L.css,11qour3ND0L.css,116t+WD27UL.css,11gKCCKQV+L.css,11061HxnEv
L.css,11oHt2HYxnL.css,01j2JE3j7aL.css,11JQtnL-6eL.css,21KA2rMsZML.css,11jtXRmppwL.css,0114z6bAEoL.css,21uwtfqr5aL.css,11QyqG8yi
```

Hide solution

In [30]:
```python
print(list(soup.html.parent))
print("The BeautifulSoup object has children")
print("The beautifulSoup object has ", len(list(soup.descendants)), "childrens")
```

```
['html', <html class="a-no-js" data-19ax5a9jf="dingo" lang="en-au"><!-- sp:feature:head-start -->
<head><script>var aPageStart = (new Date()).getTime();</script><meta charset="utf-8"/>
<!-- sp:end-feature:head-start -->
<!-- sp:feature:csm:head-open-part1 -->
<!-- sp:end-feature:csm:head-open-part1 -->
<!-- sp:feature:cs-optimization -->
<meta content="on" http-equiv="x-dns-prefetch-control"/>
<link href="https://images-fe.ssl-images-amazon.com" rel="dns-prefetch"/>
<link href="https://m.media-amazon.com" rel="dns-prefetch"/>
<link href="https://completion.amazon.com" rel="dns-prefetch"/>
<!-- sp:end-feature:cs-optimization -->
<!-- sp:feature:csm:head-open-part2 -->
<!-- sp:end-feature:csm:head-open-part2 -->
<!-- sp:feature:aui-assets -->
<link href="https://m.media-amazon.com/images/I/11EIQ5IGqaL._RC|01ZTHTZObnL.css,410yLeQZHKL.css,31OSFXVtM5L.css,013z33uKh2L.cs
s,017DsKjNQJL.css,0131vqwP5UL.css,41EWOOlBJ9L.css,11TIuySqr6L.css,01ElnPiDxWL.css,11fJbvhE5HL.css,01Dm5eKVxwL.css,01IdKcBuAdL.c
ss,01y-XAlI+2L.css,21P6CS3L9LL.css,01oDR3IULNL.css,41Js1DVdbVL.css,01XPHJk60-L.css,01S0vRENeAL.css,21IbH+SoKSL.css,11MrAKjcAKL.
css,21fecG8pUzL.css,11a5wZbuKrL.css,01CFUgsA-YL.css,31pHA2U5D9L.css,11qour3ND0L.css,116t+WD27UL.css,11gKCCKQV+L.css,11061HxnEv
L.css,11oHt2HYxnL.css,01j2JE3j7aL.css,11JQtnL-6eL.css,21KA2rMsZML.css,11jtXRmppwL.css,0114z6bAEoL.css,21uwtfqr5aL.css,11QyqG8yi
```

> Now we can navigate the whole tree. However, remember that we are only interested in some specific information on the page and not all the tags, we will see in the next notebook how to access certain elements without browsing the entire HTML structure.

✖ Unvalidate