HTML and Page Scraping Guided Exercise

In this guided exercise, we are going to look at a few of the programming libraries you are going to need for the first assessment. We are going to cover:

- HTML
- RobotParser
- Python Requests and Beautiful Soup

HTML

HTML is a markup language where content can be represented in a manner which Web browsers can understand. It is distinct from the *style* of the display, which is dealt with by Cascading Style Sheets (CSS). It is composed of a series of *tags* and attributes. The following Python variable represents an HTML document, which is structured as follows:

In [1]:

```
# Make sure that you run this code as this will be interrogated later in this exerci
html = """
        <html>
            <head>
                <!--Metadata about the document goes within the <head> tag, includir
                encoding-->
                <title>Example HTML Structure</title>

                <style type="text/css">
                    body{
                        font-size:12pt;
                    }
                    table{
                        border: solid 1px black;
                    }
                </style>

                <link href="/style.css" rel="stylesheet">

                <meta charset="utf-8">
                <meta name="viewport" content="width=device-width, initial-scale=1.0
            </head>

            <body>
                <div id="main" class="classy">
                    <h1>The Main Content of the Document</h1>

                    <p>This is some content inside a &lt;p&gt; tag.  It is normal wr
                    within it.  For example, to emphasise a word I would use <em>the
                    onto a new ine I could use the &lt;br /&gt; tag.
                    </p>

                    <table>
                        <tr>
                            <th>Table heading cell<br></th><th>Another heading</th>
                        </tr>
                        <tr>
                            <td>Normal table cell and <br></td>
                            <td>This cell has a link going <a href="#">Somewhere</a>
                        </tr>
                    </table>

                </div>

                <div id="another_div" class="classy">
                    <ul>
                        <li>An unordered (numbered) list item</li>
                        <li>And another unordered list item</li>
                    </ul>
                    <ol>
                        <li>This list item is ordered</li>
                        <li>So is this one</li>
                    </ol>
                    <table>
                        <tr>
                            <th>Table 2<br></th><th>A second heading</th>
                        </tr>
                        <tr>
                            <td>Cool content and <br></td>
```

```
                        <td>This cell has a link going <a href="#">SomewhereElse
                    </tr>
                </table>
            </div>
        </body>
    </html>
    """
```

Attributes

Several of the cells also have attributes, which is the quoted text within the opening tag. There are three attributes which are important to know:

- `href=""` is used within an anchor or <a> tag and represents the hyperlink to another resource such as another page. So `<a href="http://example.com">Destination Link</a>` will have the text "Destination Link" which when clicked on it will go to http://example.com (http://example.com)
- `id` represents the ID of a element. This should be unique within a page. It is often used for styling, but can be very useful for pagescraping, because it can be used to identify an element of interest
- `class` is similar to `id`, but it is used by more than one of the instances of a particular tag, but more specific than a general style.
- There are also some `<meta>` tags, which simply provide metadata about the document itself. These provide information about the document itself. These usually have an attribute `name=""` and `content=""`. Full details of the `<meta>` tags can be found at w3schools (https://www.w3schools.com/tags/tag_meta.asp)

Test Yourself

In the cell below, write an anchor tag which links to http://southamptondata.science (http://southamptondata.science) with the text "Data Science!" inside it

YOUR HTML HERE

Here is a link: 이건 비밀이야~! (http://southamptondata.science) 네이버임~! (http://naver.com)

There's a lot going on in that HTML snippet, so let's break it down:

Tags

Each tag of the form <tagname> represents a HTML element. Usually, the tags are closed with a tag of the same name but including / after the opening bracket, for example </tagname>, like `<body></body>`. These tags can have other tags and content inside them. Notice that `<body>` has a `<div>` tag inside it. `<div>` doesn't do much, but merely acts as a logical area of the document which can be referred to later (e.g., for styling).

For the most part, the tags represent some sort of content. A `<h1>` tag would represent a heading, with `<h2>` down to `<h6>` being subheadings. You can see a `<p>` tag - which stands for paragraph, which opens and closes such that everything inside it represents a single paragraph.

There is a `<table>` tag as well, which represents a table. Inside this tag, you will notice that there are many other tags inside that representing rows and cells.

Although most tags follow the format of opening and closing tags, some are different - referred to as "self closing" tags. These don't need to have any content inside them, but represent some part of the document. The most common example is the breakline `<br />` tag, which moves content on to the next row.

There are also special tags, which are used for different purposes:

- Comments, written as `<!-- A useful comment here -->` do not count as content, but rather are there to document the code.
- The `<style>` tag means that anything inside that will be a stylesheet, which will almost always be in CSS. This allows separation of the content (the rest of the HTML document) and the styling (this tag). It is also possible to link to CSS in an external file. We don't cover CSS in this course, but check out this CSS tutorial (https://www.w3schools.com/css/default.asp) if you are interested.
- The `<script>` tag (not included in this document) includes some form of scripting done by the browser. This will almost always be Javascript.

Test Yourself

In the next cell, try and write a paragraph tag with some text inside it including at least one breakline.

N.B. make sure that 'Markdown' format has been selected and to double click the cell if you want to edit the text.

here is my mind

Scraping With Requests and Beautiful Soup

Python has got its own libraries for dealing with HTTP, but at least historically, they required a lot of low level effort to get them to work properly. The Python requests library abstracts a lot of that away, and makes it relatively easy to simply download a page. The syntax is simple. To get a page, all you need to do is say which request verb you want (usually `get`) and write it as follows:

N.B. The version of Python Requests running on this server is 2.11.1, which was released in August 2016. This makes a change to the function which processes the headers passed to any function. Previously, it would allow any type, but the previous verion to this made a security improvement that any value has to be a string.

For example, you would put in `headers = {'header-name': '2'}`

In [4]:

```
# You don't need to write anything here
import requests
r = requests.get('http://southamptondata.science')
r.status_code

# What does the returned code mean?
# 웹사이트 접속 성공하면 200 반환
```

Out[4]:

```
200
```

This produces a `Response` object. It has many different properties, but the important ones for our purposes are:

- `r.text` The text of the body of the response, in this case the HTML of the Web page.

- `r.status_code` An integer which represents the success of the request. For example, `200` means sucess and `404` means page not found. See the full list of status codes on Wikipedia (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

It is not only HTML that the `requests` library can accept, but it can make any HTTP request. Notably, there is a `.json()` function which will give the JSON associated with any request.

We can also add more detail to the request by adding further parameters to the request. You can add a `params=` and `headers=` parameter, which allow you to edit the parameters and headers respectively. Check out the Quickstart (http://docs.python-requests.org/en/master/user/quickstart/) and Advanced Usage (http://docs.python-requests.org/en/master/user/advanced/) for some examples of more of the things you can do.

Write your own request to go to the GitHub homepage (https://github.com). Manually check on whether you are allowed to visit according to the `robots.txt` file, generate the request, and then inspect the response that you get. You can use the function `vars` to give details of any Python object, taking the variable as a parameter, or you can get individual fields from the response such as `status_code`, `content` or `headers`.

In [7]:

```
# YOUR CODE HERE
r = requests.get('http://github.com')
#이 웹사이트에 requests.get 응답이 변수 r에 지정 되었음.
r.content
r.headers
#콘텐츠 가져오기, 해더가져오기
r.status_code

print(r.headers)
#첫번째 페이지 헤더 정보 (수집한 정보 전체의 헤더)
print(r.request.headers)
#보낸 요청의 헤더 정보
```

```
{'Accept-Ranges': 'bytes', 'Content-Encoding': 'gzip', 'X-XSS-Protecti
on': '1; mode=block', 'Referrer-Policy': 'origin-when-cross-origin, st
rict-origin-when-cross-origin', 'Strict-Transport-Security': 'max-age=
31536000; includeSubdomains; preload', 'Server': 'GitHub.com', 'Vary':
'X-PJAX, Accept-Encoding, Accept, X-Requested-With, Accept-Encoding',
'X-GitHub-Request-Id': 'AF86:68F8:27970B:2F0963:600AD878', 'Date': 'Fr
i, 22 Jan 2021 13:51:48 GMT', 'Content-Security-Policy': "default-src
'none'; base-uri 'self'; block-all-mixed-content; connect-src 'self' u
ploads.github.com www.githubstatus.com collector.githubapp.com api.git
hub.com github-cloud.s3.amazonaws.com github-production-repository-fil
e-5c1aeb.s3.amazonaws.com github-production-upload-manifest-file-7fdce
7.s3.amazonaws.com github-production-user-asset-6210df.s3.amazonaws.co
m cdn.optimizely.com logx.optimizely.com/v1/events wss://alive.github.
com github.githubassets.com; font-src github.githubassets.com; form-ac
tion 'self' github.com gist.github.com; frame-ancestors 'none'; frame-
src render.githubusercontent.com; img-src 'self' data: github.githubas
sets.com identicons.github.com collector.githubapp.com github-cloud.s
3.amazonaws.com *.githubusercontent.com customer-stories-feed.github.c
om spotlights-feed.github.com; manifest-src 'self'; media-src github.g
ithubassets.com; script-src github.githubassets.com; style-src 'unsafe
-inline' github.githubassets.com; worker-src github.com/socket-worker-
5029ae85.js gist.github.com/socket-worker-5029ae85.js", 'Content-Typ
e': 'text/html; charset=utf-8', 'X-Content-Type-Options': 'nosniff',
'X-Frame-Options': 'deny', 'Cache-Control': 'max-age=0, private, must-
revalidate', 'Transfer-Encoding': 'chunked', 'ETag': 'W/"6bbcc58f01339
5d64bc20974c495c305"', 'Expect-CT': 'max-age=2592000, report-uri="http
s://api.github.com/_private/browser/errors"', 'Set-Cookie': '_gh_sess=
Uen4fPw2brpK9rsIb1Frer7OtPGjRrwsYzri4TI4GbJ9maxrV2j7J%2FP%2BxKyidLUgAb
tNfoD6r1DPG90BsfKLKsqQAmujIWPdma3u68yaItYVdNEQfCoRmY%2FcDa5d8RSuR76KaV
7zmSGvs%2FNo9MLwfBTNes6YHRAeu7muvDrnyGk13WW%2BhDTPQkE62Jj3ODyBIPVQ3TlV
Q8XPzTjACdLs%2BD55P8Qe4QuvOeQ5rQqK3zkoTsFvFMDzhb8O7022OsTsGMH62L0pPoon
jZFUXhN1AQ%3D%3D--sEgL0ylgQekcbeOJ--P%2Fq1FUleUYkfOELmpvCN%2Bw%3D%3D;
Path=/; HttpOnly; Secure; SameSite=Lax, _octo=GH1.1.283055964.16113235
12; Path=/; Domain=github.com; Expires=Sat, 22 Jan 2022 13:51:52 GMT;
Secure; SameSite=Lax, logged_in=no; Path=/; Domain=github.com; Expires
=Sat, 22 Jan 2022 13:51:52 GMT; HttpOnly; Secure; SameSite=Lax', 'Stat
us': '200 OK'}
{'User-Agent': 'python-requests/2.11.1', 'Accept-Encoding': 'gzip, def
late', 'Accept': '*/*', 'Connection': 'keep-alive'}
```

An HTTP header (https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) is part of the HTTP protocol, which defines the way in which a particular HTTP session will work. For example, a client could declare itself to be a certain type of browser by using the `User-Agent` header, and this was previously used by servers as a means of deciding how to serve particular content, which has a long and complex history (http://webaim.org/blog/user-agent-string-history/).

See how the Python requests library declares itself, and in the cell below try and change some of the headers in the request, using the `headers` parameter to the request.

In [8]:

```
# headers={'User-Agent': 'New '}
r = requests.get('http://178.62.90.232/')
print(r.request.headers)
```

```
{'User-Agent': 'python-requests/2.11.1', 'Accept-Encoding': 'gzip, def
late', 'Accept': '*/*', 'Connection': 'keep-alive'}
```

In [ ]:

```
# YOUR CODE HERE
```

Robots.txt

To check whether you are allowed to access Web pages using a scraper, make sure you check **`robots.txt`**. Python has a class called `RobotFileParser` (https://docs.python.org/3.0/library/urllib.robotparser.html) which does exactly this. It is quite simple to use, and lets you know whether you can read the page.

Look at the documentation and check whether you can read different parts of the BBC (http://www.bbc.co.uk) website.

In [10]:

```
# You don't need to write anything here
from urllib.robotparser import RobotFileParser
rp = RobotFileParser()
rp.set_url('http://bbc.co.uk/robots.txt')
#파일 경로 살펴보기
rp.read()
#RobotFileParser의 read 함수를 사용
rp.can_fetch("*", 'http://bbc.co.uk/news')
#모든 정보를 다 가져와 달라고 요청
#트루면 정보를 가져가도 된다.
```

Out[10]:

```
True
```

Parsing HTML

To parse the HTML, we will use the Beautiful Soup (https://www.crummy.com/software/BeautifulSoup/bs4/doc/) library. This uses a class **`BeautifulSoup`** which accepts a **string** as a parameter (the HTML) and allows us to search through it - in other words Beautiful Soup is a Python library for pulling data out of HTML and XML files. You can look through the documentation at the link for more of the details, but the basic important parts of it can be seen in the next cell:

N.B. This uses version 4 of Beautiful Soup. The previous version did not require you to specify a parser as one of the parameters, whereas this version does. *For HTML you should use 'lxml', and for XML you should use 'xml'*.

In [11]:

```
# N.B. Make sure you have run the code in the first code cell with the HTML in it be
# The code here relies on the `html` variable.

# You don't need to write anything here
from bs4 import BeautifulSoup

# Generate the BeautifulSoup instance, and make use of the LXML parser
soup = BeautifulSoup(html, 'lxml')

# Find the first instance of the <table> element
table = soup.find('table')
#테이블에 모든 인스턴스를 가져옵니다 -> find 함수
print(table)


# Find all instances of <tr> within the <table> element we just found
# We could do that from any point in the HTML tree.
rows = soup.find_all('tr')
for r in rows:
    print(r.text)
#tr에 r에 있는 인스턴스와 관련된 텍스트 인쇄

links = soup.find_all('a')
for l in links:
    print('link texts', l.text)
#link texts 헤딩임, 하이퍼링크 자체가 아니라 links의 텍스트를 요청
print('\n')


# Find a table in the element with ID another_div
table = soup.find(id='another_div').find('table')
print(table)

print('\n')

lists = soup.find_all('ul')
for lis in lists:
    print(lis.text)
#정렬되지않은 리스트


lister = soup.find_all('ol')
for lisp in lister:
    print(lisp.text)
#정렬된 리스트

print('\nthis is some text between tds')
cols = soup.find_all('td')
for c in cols:
    print('\ncolumn text', c.text)


# Get the value of an attribute
soup.find('meta')['charset']
#메타데이터를 살펴보고 문자세트 charset을 찾기
```

```
<table>
<tr>
```

```
<th>Table heading cell<br/></th><th>Another heading</th>
</tr>
<tr>
<td>Normal table cell and <br/></td>
<td>This cell has a link going <a href="#">Somewhere</a></td>
</tr>
</table>

Table heading cellAnother heading


Normal table cell and
This cell has a link going Somewhere


Table 2A second heading


Cool content and
This cell has a link going SomewhereElse

link texts Somewhere
link texts SomewhereElse


<table>
<tr>
<th>Table 2<br/></th><th>A second heading</th>
</tr>
<tr>
<td>Cool content and <br/></td>
<td>This cell has a link going <a href="#">SomewhereElse</a></td>
</tr>
</table>



An unordered (numbered) list item
And another unordered list item


This list item is ordered
So is this one


this is some text between tds

column text Normal table cell and

column text This cell has a link going Somewhere

column text Cool content and

column text This cell has a link going SomewhereElse
```

Out[11]:

```
'utf-8'
```

In general, the `.find()` method will give a `Tag` object, with all the properties of a HTML tag which can be

explored, and the `find_all()` method will give a list of `Tag` objects (if they exist).

Get some HTML from a page, and experiment using this library. Aim to find the following:

- All anchor (i.e., `<a>`) tags
- The names of the id and class of the `<div>` tags
- The text within some `<td>` tags

In [12]:

```
# YOUR CODE HERE
print(soup.find_all('a'))
for d in soup.find_all('div'):
    print(d['id'])
    print(d.attrs['class'])
for t in soup.find_all('td'):
    print(t.text)
```

```
[<a href="#">Somewhere</a>, <a href="#">SomewhereElse</a>]
main
['classy']
another_div
['classy']
Normal table cell and
This cell has a link going Somewhere
Cool content and
This cell has a link going SomewhereElse
```

In [ ]:

```

```