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 [2]:

```
# 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, including St
            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 writin
                within it.  For example, to emphasise a word I would use <em>the &lt
                onto a new ine I could use the &lt;br /&gt; tag.
                </p>

                <table>
                    <tr>
                        <th>Table heading cell</th><th>Another heading</th>
                    </tr>
                    <tr>
                        <td>Normal table cell</td><td>This cell has a link going <a
                    </tr>
                </table>

            </div>

            <div id="another_div" class="classy">
                <ul>
                    <li>A list item</li>
                    <li>And another list item</li>
                </ul>
                <ol>
                    <li>This list item is numbered</li>
                    <li>So is this one</li>
                </ol>
                <table>
                    <tr>
                        <th>Table 2</th><th>A second heading</th>
                    </tr>
                    <tr>
                        <td>Cool content</td><td>This cell has a link going <a href=
                    </tr>
                </table>
            </div>
        </body>
```

```
    </html>
"""
```

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.

YOUR HTML HERE

```
<p>
This is some content inside a &lt;p&gt; tag.  It is like normal writing.
For example, to emphasise a word I would use the &lt;em&gt;tag&lt;/em&gt;
or to move onto a new line I could use the &lt;br /&gt; tag.
And then keep writing like this
</p>
```

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: Data Science! (http://southamptondata.science)

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 incredibly 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, any variable type was allowed in the header e.g. integers, floats, etc., but the version previous to 2.11.1 implemented a security improvement such that the value has to be a string.

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

Below is an example of using the Python requests library. Running the code cell will output a code - for your own interest why not look up this code online. What does it mean and what other code are there?

In [3]:

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

Out[3]:

```
'<!doctype html>\r\n<html lang="en" class="no-js">\r\n<head><meta char
set="utf-8" /><meta http-equiv="X-UA-Compatible" content="IE=edge" /><
meta name="language" content="en" /><meta name="generator" content="Is
le Interactive Ltd" /><meta name="designer" content="Isle Interactive
Ltd" /><meta name="viewport" content="initial-scale=1.0, width=device-
width" /><meta name="application-name" content="CEG Digital Southampto
n" /><meta name="msapplication-TileColor" content="#ffffff" /><meta na
me="msapplication-TileImage" content="/_img/tile.png" />\r\n        <m
eta name="robots" content="index, follow" />\r\n    <meta name="revisi
t-after" content="14 days" /><meta name="google-site-verification" con
tent="LmoHnAS6r540RRceWUi8Hil9zxxhUZC1RER9NM2SRD8" />\r\n\t<!-- Google
Tag Manager -->\r\n    <script>(function(w,d,s,l,i){w[l]=w[l]||[];w
[l].push({\'gtm.start\':\r\n    new Date().getTime(),event:\'gtm.js
\'});var f=d.getElementsByTagName(s)[0],\r\n    j=d.createElement(s),d
l=l!=\'dataLayer\'?\'&l=\'+l:\'\';j.async=true;j.src=\r\n    \'http
s://www.googletagmanager.com/gtm.js?id=\'+i+dl;f.parentNode.insertBefo
re(j,f);\r\n    })(window,document,\'script\',\'dataLayer\',\'GTM-5XFK
B6X\');</script>\r\n    <!-- End Google Tag Manager -->\r\n\r\n\r\n
```

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). Using the `robots.txt` file which Huw discussed in one of his videos manually check whether you are allowed to visit the github site 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 [4]:

```
# YOUR CODE HERE
r = requests.get('https://github.com')
#r.content
r.headers
#r.status_code
```

Out[4]:

```
{'Date': 'Wed, 27 Jan 2021 03:39:18 GMT', 'Expect-CT': 'max-age=259200
0, report-uri="https://api.github.com/_private/browser/errors"', 'Stat
us': '200 OK', 'X-Content-Type-Options': 'nosniff', 'Accept-Ranges':
'bytes', 'X-Frame-Options': 'deny', 'Set-Cookie': '_gh_sess=lFvZHeyGCn
J16HzbzAsz%2BASN%2F0ACMfnDrSuklCSYaEoPS6RhsCdP2fhr1BoDO8QZGap2NPbiM7lc
LZwWXWAfUl9D%2Bj54l%2BkWUZbYhwX5KXdhszttrSWtyl1vqjwzAd9QF95LhutYURGXri
byqge6u7EnIm6t9p5GpSpybE5VfBaNsueroDRbgOUS3xTzmLLDhMdlWgYGU0bdLvAgzsnp
pe0lvAkIi3%2FBDpM67VBRquzi9HcaldsrlQ76gvji1Mk40hxeTx1z7lcBARLjk%2Fxc5
g%3D%3D--VAC0hvhZI%2BEHoDN1--xW0RGkY%2FI4HTjLj6WK0LYg%3D%3D; Path=/; H
ttpOnly; Secure; SameSite=Lax, _octo=GH1.1.296818112.1611718765; Path
=/; Domain=github.com; Expires=Thu, 27 Jan 2022 03:39:25 GMT; Secure;
SameSite=Lax, logged_in=no; Path=/; Domain=github.com; Expires=Thu, 27
Jan 2022 03:39:25 GMT; HttpOnly; Secure; SameSite=Lax', 'Content-Typ
e': 'text/html; charset=utf-8', 'Cache-Control': 'max-age=0, private,
must-revalidate', 'Content-Encoding': 'gzip', 'ETag': 'W/"ba0468956fe3
7dfc511448afc3efe4df"', 'Referrer-Policy': 'origin-when-cross-origin,
strict-origin-when-cross-origin', 'Vary': 'X-PJAX, Accept-Encoding, Ac
cept, X-Requested-With, Accept-Encoding', 'Strict-Transport-Security':
```

An HTTP header (https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) is part of the HTTP protocol, which define 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/) (interesting reading).

Look at 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 [5]:

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

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

In [6]:

```
# YOUR CODE HERE
# An example of how to modify a header so that the scraper is disguised as a Firefox

r = requests.get('http://178.62.90.232',
                 headers={'User-Agent': 'Mozilla/5.0 (Android 4.4; Tablet; rv:41.0)
print(r.request.headers)
```

```
{'Connection': 'keep-alive', 'User-Agent': 'Mozilla/5.0 (Android 4.4;
Tablet; rv:41.0) Gecko/41.0 Firefox/41.0', 'Accept': '*/*', 'Accept-En
coding': 'gzip, deflate'}
```

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 [7]:

```
# You don't need to write anything here
from urllib.robotparser import RobotFileParser
rp = RobotFileParser()
rp.set_url('http://bbc.co.uk/robots.txt')
rp.read()
rp.can_fetch("*", 'http://bbc.co.uk/news')
```

Out[7]:

```
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. 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 the second parameter, if the text is **HTML** you should use **'lxml'**, and for **XML** you should use **'xml'**.

In [16]:

```
# 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')
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 = table.find_all('tr')
for r in rows:
    print(r.text)

links = soup.find_all('a')
for l in links:
    print('link texts', l.text)

print('\n')


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

print('\nthis is some text between tds')
cols = table.find_all('td')
for r in cols:
    print(r.text)


# Get the value of an attribute
soup.find('meta')['charset']
```

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

Table heading cellAnother heading


Normal table cellThis cell has a link going Somewhere

link texts Somewhere
link texts Somewhere
```

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

this is some text between tds
Cool content
This cell has a link going Somewhere

Out[16]:
'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 [19]:

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

In [22]:

```
url = 'http://138.68.148.20/data/west_midlands/cannock_chase'
res = requests.get(url)
soup = BeautifulSoup(res.content, 'xml') #res에서 얻은 url을 xml 파일 가져오기
IC = 0
BN = ""

for c in soup.find("ItemCount"):
        IC = c

for n in soup.find("BusinessName"):
        BN = n

dicty = { 'amount_of_records' : int(IC), 'first_business' : BN }

print(dicty)
```

```
{'amount_of_records': 731, 'first_business': '1st Choice Pizza/Fish &
Chips'}
```

In [33]:

```
j = soup.ItemCount.string
j1 = int(j1)

k = soup.BusinessName.string

dic = { 'amount_of_records' : j1, 'first_business' : k}
print(dic)
```

```
1st Choice Pizza/Fish & Chips
{'amount_of_records': 731, 'first_business': '1st Choice Pizza/Fish &
Chips'}
```

In [35]:

```
busn = soup.find("BusinessName").text
itc = soup.find("ItemCount").text

dictt = { 'first_business' : busn, 'amount_of_records' : int(itc) }
print(dictt)
```

```
{'amount_of_records': 731, 'first_business': '1st Choice Pizza/Fish &
Chips'}
```

In [ ]: