ST115 Managing and Visualising data

Lecture 8: Collecting data from the Internet

2022/23 Lent Term

Today's plan

- Internet and the web
- Collecting data from the Internet
    - Web scraping

Self-study Python materials

Useful Python functionality (1): f-string

Example: To create a string `Hello, x!`, with x someone's name stored in a variable.

- By string concatenation:

In [1]:
```
name = 'Harry'
'Hello, '+name+'!'
```

Out[1]:

```
'Hello, Harry!'
```

- By f-string:

In [2]:
```
f'Hello, {name}!'
```

Out[2]:

```
'Hello, Harry!'
```

`{name}` is replaced by the value of the variable. Note you need the `f` in front of the quotation. The following does not work:

In [3]:
```
'Hello, {name}' # f is missing
```

Out[3]:

```
'Hello, {name}'
```

Useful Python functionality (2): split string

Sometimes a string can be very long. You can split it into pieces by the use of `\`. Examples:

In [4]:

```
s = 'This is a very very very long string, too long to display ' \
'in one line so we are going to split it into half and here ' \
'we are showing you how to do it with the use of \\.'
print(s)
```

```
This is a very very very long string, too long to display in one line so we are going
to split it into half and here we are showing you how to do it with the use of \.
```

Internet

The Internet is a huge global network of computers all connected together.

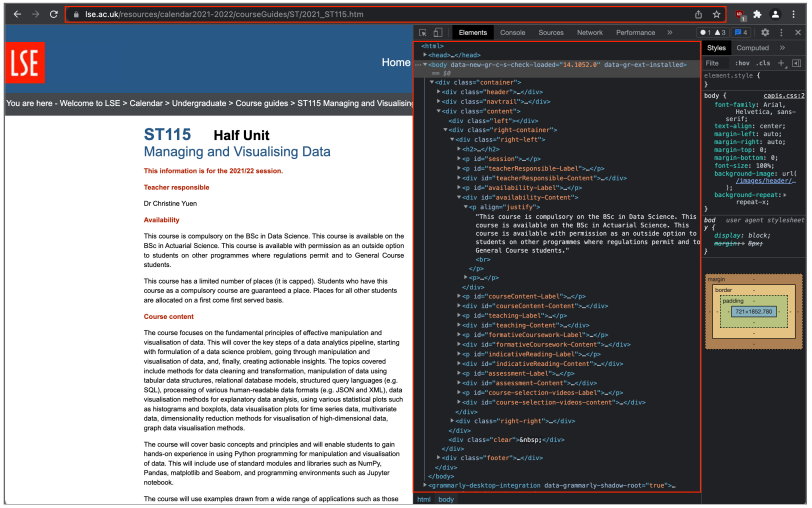
- Through the Internet, people can share information and communicate from anywhere with an Internet connection
    - The world wide web and email are some applications running on top of the Internet

World wide web

World wide web (WWW) is an information system of web pages and other web resources accessible over the Internet.

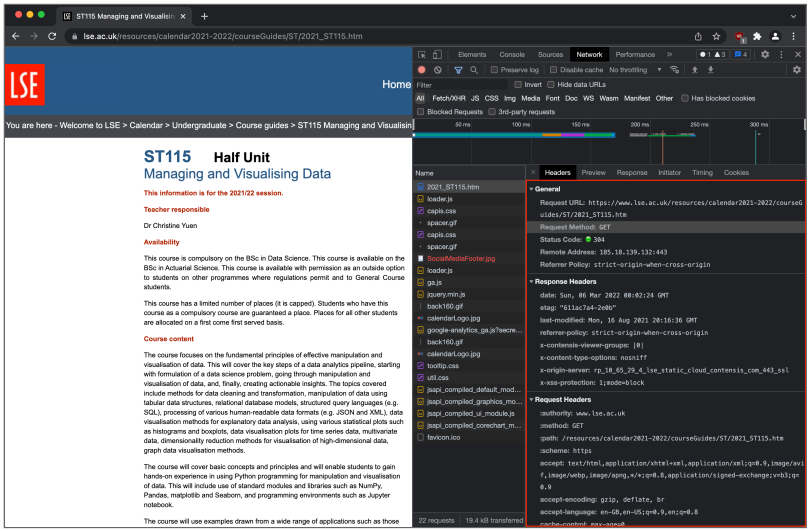
- It is commonly known as the Web
- Three important components:
    - URL: addressing scheme to find resources
    - HTTP: protocol on how web browsers and web servers communicate
        - Protocol: set of standard conventions that the world has agreed upon for computers to communicate with
    - HTML: a markup language to format pages containing *hypertexts*
        - Hypertext: text displayed on a computer display that links to other text the reader can immediately access

Example: URL and html



You can see the html code by right clicking on a web page and click "Inspect Element" (if you are using safari) or "Inspect" (if you are using chrome)

Example: HTTP



URL

URL (Uniform Resource Locator, or "web address") can be considered as an address of a given unique resource.

- Example: **https://www.lse.ac.uk/resources/calendar2021-2022/courseGuides/ST/2021_ST115.htm**
- URLs occur most commonly to reference web pages (http) but are also used for file transfer (ftp), email (mailto), etc
    - e.g. **mailto:statistics@lse.ac.uk**

URL (continue)

A typical URL could have the form: `<scheme>://<host>:<port>/<path><;parameters><?query><#fragment>`.

- Scheme (e.g., http, https, mailto, ftp)
- Location of the server, e.g. a hostname or IP address
- [optional] port: communication endpoint
- Path to the resource on the server
- [optional] parameters (;something), query string (?something), fragment identifier (#something)

URL example 1

https://www.lse.ac.uk/resources/calendar2021-2022/courseGuides/ST/2021_ST115.htm

- `https` : scheme
- `www.lse.ac.uk` : hostname which can help with identifying the location of the server
- `resources/calendar2021-2022/courseGuides/ST/2021_ST115.htm` : path to the resource on the serve

URL example 2

http://216.58.194.196/search?q=lse

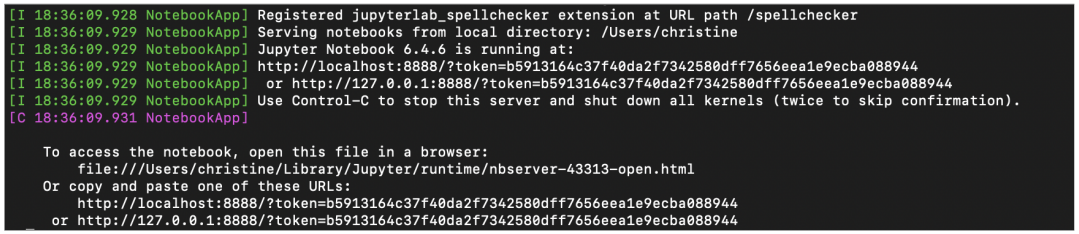
- `http` : scheme
- `216.58.194.196` : IP address (Internet Protocol address) of Google
- `?q=lse` : query string

We will see more examples of the query string when we talk about API.

URL example 3

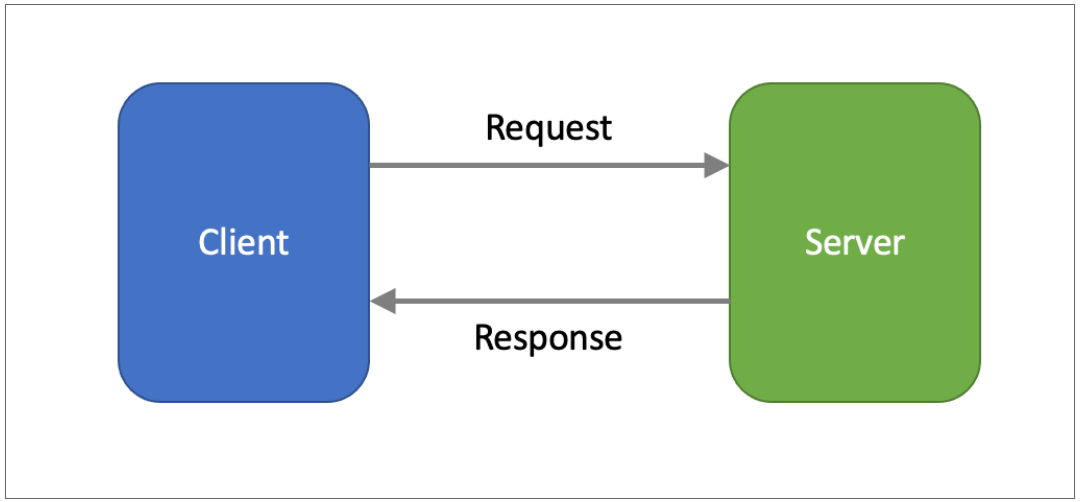
http://127.0.0.1:8888/notebooks/ST115/week_08_webscraping/lecture_8.ipynb#URL-example-3
(this jupyter notebook on my computer)

- `http` : scheme
- `127.0.0.1` : IP address of the current device (or *localhost*)
- `8888` : port
- `notebooks/ST115/week_08_webscraping/lecture_8.ipynb` : path to the resource on the serve
- `#URL-example-3` : fragment identifier



Servers

- Server is a piece of computer hardware or software that provides functionality for *clients*
    - Client: a piece of computer hardware or software that accesses a service made available by a server
- Communications between server and client:



Behind the scene when loading a web page

- After entering a URL in a browser (or clicking on some links, hitting a button, etc), your computer requests another computer (server) specified by the given URL for information
- Server decodes the request, sends back the information (response)
- Your computer interprets the response and display it

HTTP

HTTP (Hypertext Transfer Protocol) is a protocol which determines how a web page is requested and what the response looks like.

- Example of HTTP request header:

HTTP requests

- Request messages are sent by a client to a target server
- A HTTP request has the following components:
    - Request line, which includes
        - URL
        - Method (e.g. GET)
        - Protocol version
    - Headers: meta-information about a request
    - Message body (optional)

HTTP request methods

HTTP request methods indicate the desired action to be performed on the identified resource.

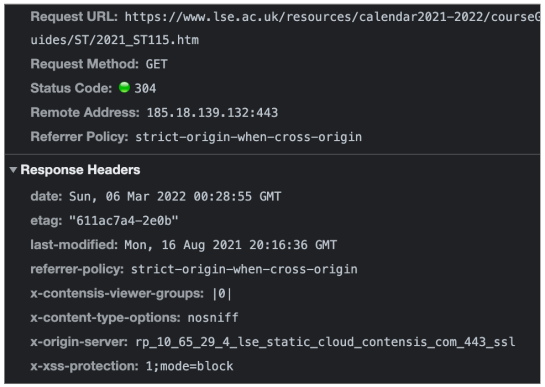
- Some example request methods:

Method	Description
GET	Retrieve information from the specified URL
HEAD	Retrieve the meta information from the header of the specified URL
POST	Send the attached information for appending to the source URL
PUT	Send the attached information for replacing the resource at the specified URL
DELETE	Delete the information identified by the URL

A GET request is sent when you enter a url in browser. In this lecture we only consider GET as we focus on retrieving information.

HTTP response

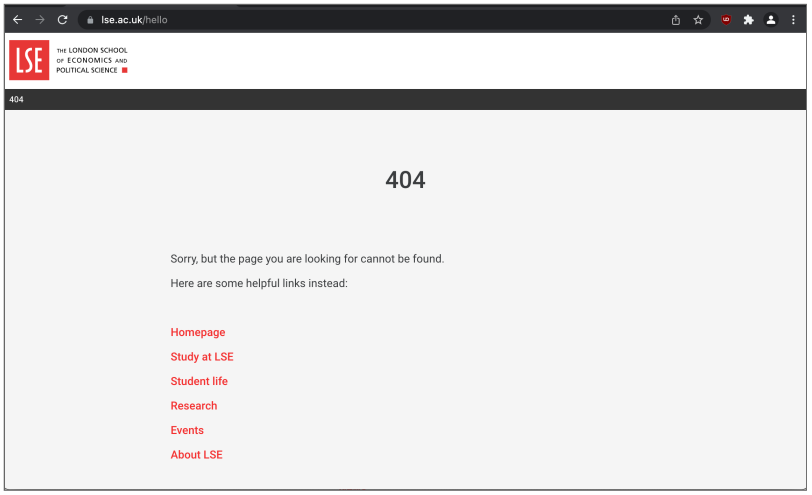
- Response message is sent by a server to a client as a reply to its former request.
- HTTP response has the following components:
    - Status line, which includes
        - Status code (numerical, e.g. 404) with the textual reason phrase (e.g. Not Found)
        - Protocol version
    - Headers
    - Message body (optional)



HTTP status code

Status Code	Description
1xx	Informational response e.g. server is processing the request
2xx	Success. The action requested by the client was received, understood, and accepted
3xx	Redirection e.g. when requested resource has moved
4xx	Client error. Examples: 401 *unauthorised* 404 *page not found* 429 *too many requests*
5xx	Server error

Example: 404 Not found



Sending request in Python

In Python, we can use the `requests` module, which allows users to send HTTP requests and get the response easily.

In [4]:
```
import requests
url = 'https://www.lse.ac.uk/resources/calendar2022-2023/courseGuides/ST/2022_ST115.htm'
r = requests.get(url)
```

In [5]:
```
r.status_code, r.reason
```

Out[5]:

```
(200, 'OK')
```

In [6]:
```
r.headers
```

Out[6]:

```
{'server': 'nginx/1.10.3 (Ubuntu)', 'date': 'Sun, 05 Mar 2023 19:02:23 GMT', 'content-
type': 'text/html', 'last-modified': 'Mon, 16 Jan 2023 14:40:20 GMT', 'transfer-encodi
ng': 'chunked', 'etag': 'W/"63c561d4-30de"', 'x-contensis-viewer-groups': '|0|', 'cont
ent-encoding': 'gzip', 'x-origin-server': 'rp_10_65_29_4_lse_static_cloud_contensis_co
m_443_ssl', 'x-xss-protection': '1;mode=block', 'x-content-type-options': 'nosniff',
'referrer-policy': 'strict-origin-when-cross-origin', 'connection': 'close'}
```

In [7]:
```
r.text # see the content of the response (here is HTML code)
```

Out[7]:

```
'<html>\n<head>\n<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1">\n<title>ST115 Managing and Visualising Data</title>\n<meta content="Calendar web e
ditor" name="dc.creator">\n<meta content="calendar@lse.ac.uk" name="dc.creator.email">
\n<meta content="Academic Registrar&#39;s Division" name="lse.pagesource">\n<link rel
="stylesheet" type="text/css" href="/css/capis.css">\n<link href="../../Default.htm" r
el="start" title="Calendar">\n<link href="../../undergraduate.htm" rel="prev" title="U
ndergraduate">\n<link href="../../courseGuides/undergraduate.htm" rel="prev" title="Co
urse guides">\n<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.
min.js" type="text/javascript"></script><script src="https://www.gstatic.com/charts/lo
ader.js" type="text/javascript"></script><script type="text/javascript">\n\n        fu
nction buildChart() { // made sure function is loaded before so that google chart\'s c
allback needs it\n\n            var examsContentId ;\n            var examsData ;\n
var examsChart ;\n\n            function loadChartData(id, nr) {\n                var
data = new Array();\n                var heading = new Array();\n                $(\'#
\' + id + \' > table th\').each(function() {\n                    heading.push($(thi
s).html().replace(\'<br>\', \' \'));\n                });\n                data.push(h
eading);\n                var line = null;\n                $(\'#\' + id + \' > table
> tbody > tr:visible > td\').each(function(index) {\n                    if (index % n
r == 0) {\n                        line = new Array();\n                        line.p
ush($(this).text());\n                    }\n                else {\n
if ($(\'table\', $(this)).length == 0) {    // ignore nested table.\n
line.push(parseFloat($(this).text()));    // convert to number.\n
data.push(line);\n                    }\n                }\n
});\n\n                return data;\n            };\n\n            // chart for exam c
lassification.\n            examsContentId = \'exams-Content\';\n            if ($(\'#
\' + examsContentId).length > 0) {\n                examsData = google.visualization.a
rrayToDataTable(loadChartData(examsContentId, 2));\n                examsChart = new g
oogle.visualization.BarChart(document.getElementById(examsContentId + \'-Graph\'));\n
examsChart.draw(examsData, {\n                    width: 400,\n                    hei
ght: 250,\n                    colors: [\'#285886\'],\n                    tooltip: {i
sHtml: false, textStyle: {fontSize: 13, bold: false, italic: false}},\n
legend: {position: \'none\'},\n                    chartArea: {width: \'70%\', height:
\'75%\'},\n                    vAxis: {textStyle: {fontSize: 11, bold: false, italic:
false}},\n                    hAxis: {\n                        title: \'% of students
\',\n                        textPosition: \'out\',\n                        minValue:
0,\n                        maxValue: 100,\n                        titleTextStyle: {f
ontSize: 12, bold: false, italic: false}\n                    }\n                });\n
\n                $(\'#\' + examsContentId + \' table\').hide();    // hide exams data
table.\n            }\n\n        } // end of buildChart()\n\n        google.charts.loa
d(\'current\', {\'packages\':[\'corechart\']\n            , \'callback\': buildChart
// alt to .setOnLoadCallback() ensures google.charts guaranteed to have callback code
```

\n   });\n\n\n   </script><script type="text/javascript">var _gaq = _gaq || [];_
gaq.push([\'_setAccount\', \'UA-1994817-8\']); _gaq.push([\'_setDomainName\', \'lse.a
c.uk\']); _gaq.push([\'_addIgnoredRef\', \'lse.ac.uk\']); _gaq.push([\'_setAllowAnchor
\', true]); _gaq.push([\'_trackPageview\']);(function() { var ga = document.createElem
ent(\'script\'); ga.type = \'text/javascript\'; ga.async = true; ga.src = (\'https:\'
== document.location.protocol ? \'https://ssl\' : \'http://www\') + \'.google-analytic
s.com/ga.js\'; var s = document.getElementsByTagName(\'script\')[0]; s.parentNode.inse
rtBefore(ga, s); })();</script>\n</head><body> \n<div class="hea
der">\n \n<img src="/images/spacer.gif" alt
="Skip" width="1" height="1" border="0">Home |\n\t<a href="/resour
ces/help/">Help |\n\tSearch |\n   <a href="htt
p://www.lse.ac.uk/lseforyou/">LSE for You \n\t \n<div class="header-imag
e"> \n   \n \n \n<a name="n
avtrail"><img src="/images/spacer.gif" alt="Skip" width="1" hei
ght="1" border="0"> You are here - Welcome to LSE > <a hre
f="../../Default.htm">Calendar > Undergraduat
e > Course guides > ST115
Managing and Visualising Data \n \n \n
\n \n<h2>\nS
T115   Half Unit
M
anaging and Visualising Data\n</h2>\n This information is for
the 2022/23 session. \n Teacher respon
sible \n \n
\n Dr Mona Azadkia \n Availability</sp
an> \n \n This course is compu
lsory on the BSc in Data Science. This course is available on the BSc in Actuarial Sci
ence. This course is available with permission as an outside option to students on oth
er programmes where regulations permit and to General Course students.
 \n Thi
s course has a limited number of places (it is capped). Students who have this course
as a compulsory course are guaranteed a place. Places for all other students are alloc
ated on a first come first served basis. \n <s
pan>Pre-requisites \n \n<p align="justif
y"> \n Students who have no previous experience in Python are required to take an
online pre-sessional Python course from the Digital Skills Lab (https://moodle.lse.ac.
uk/course/view.php?id=7696). \n Although not a formal requirement, it is preferab
le that students have some familiarity with the basic concepts of probability and stat
istics, to the level of ST102/ST107 first 2 chapters (Data visualisation and descripti
ve statistics and probability theory). \n <spa
n>Course content \n \n<p align="justif
y"> \n The course focuses on the fundamental principles of effective manipulation
and visualisation of data. This will cover the key steps of a data analytics pipeline,
starting with formulation of a data science problem, going through manipulation and vi

sualisation of data, and, finally, creating actionable insights. The topics covered in clude methods for data cleaning and transformation, manipulation of data using tabular data structures, relational database models, structured query languages (e.g. SQL), pr ocessing of various human-readable data formats (e.g. JSON and XML), data visualisatio n methods for explanatory data analysis, using various statistical plots such as histo grams and boxplots, data visualisation plots for time series data, multivariate data, graph data visualisation methods. \n The course will cover basic concepts and pri nciples and will enable students to gain hands-on experience in using Python programmi ng for manipulation and visualisation of data. This will include use of standard modul es and libraries such as NumPy, Pandas, Matplotlib and Seaborn, and programming enviro nments such as Jupyter notebook. \n The course will use examples drawn from a wid e range of applications such as those that arise in online services, social media, soc ial networks, finance, and machine learning. The principles and methods learned will e nable students to effectively derive insights from data and communicate results to end users. \n Teaching \n \n 20 hours of lectures and 15 hours of seminars in the LT.
 \n This course will be delivered through a combination of classes, lectures and Q&A sessions totalling a minimum of 35 hours in Lent Term. This cours e includes a reading week in Week 6 of Lent Term. \n Students are required to ins tall Python on their own laptops and use their own laptops in the lectures and classe s. \n Formative coursework \n \n Students will be expected to produce 8 exercises in the LT.
 \n Weekly exercises will be given, using Python and various libraries to apply various data manipulation and visualisatio n methods to data.   \n In dicative reading \n \n \n
Essential Reading:\n
\n
W. Mckinney, Python for Data Analysis, 2nd Edition, O’Reilly 2017
\n
A. C. Muller and S. Guido, Introduction to Machine Learning with Python, O’Reilly, 2016
\n
Easley, David, and Jon Kleinberg. Networks, crowds, and markets: Reasoning about a hi ghly connected world. Cambridge university press, 2010
\n
R. Ramakrishnan and J. Gehrke, Database Management Systems, 3rd Edition, McGraw Hill, 2002
\n \n Additional Reading: \n
\n
NumPy, https://numpy.or g/
\n
Python Data Analysis Library, https://pandas.pydata.org/
\n
Matplo tlib, https://matplotlib.org
\n
Seaborn: statistical data visualization http s://seaborn.pydata.org
\n
NetworkX: Software for complex networks, http s://networkx.org
\n \n Assessment</spa n> \n \n Coursework (30%) and pr oject (70%) in the LT.
 \n Students are required to hand in solutions to 2 set s of exercises using Python each accounting for 15% of the final assessment, and hand in a report for a project (accounting for 70% of the final assessment). The project co nsists of applying data manipulation and visualisation methods to some dataset(s).

```
</div> </div>\n<div class="right-right">\n<p id="keyFacts-Label"><b> <span>Key facts</
span> </b> </p>\n<div id="keyFacts-Content">\n<p align="justify">Department: Statistic
s</p>\n<p align="justify">Total students 2021/22: 56</p>\n<p align="justify">Average c
lass size 2021/22: 29</p>\n<p align="justify">Capped 2021/22: Yes (60)</p>\n<p align
="justify">Value: Half Unit</p> </div>\n<a href="http://www.lse.ac.uk/resources/calend
ar/courseGuides/GuidelinesForCourseGuides.htm" id="ecgInstructions" target="_blank">Gu
idelines for interpreting course guide information</a><p id="course-selection-videos-L
abel"><b>Course selection videos</b></p>\n<div id="course-selection-videos-content">\n
<p>Some departments have produced short videos to introduce their courses. Please refe
r to the <a href="https://www.lse.ac.uk/resources/calendar/courseGuides/CourseIntroduc
tionVideos.htm">course selection videos index page</a> for further information.</p> </
div>\n<p id="pdam-Label"><b>Personal development skills</b></p>\n<div id="pdam-Conten
t">\n<ul>\n<li>Self-management</li>\n<li>Team working</li>\n<li>Problem solving</li>\n
<li>Application of information skills</li>\n<li>Communication</li>\n<li>Application of
numeracy skills</li>\n<li>Specialist skills</li> </ul> </div>\n</div>\n</div>\n<div cl
ass="clear"> </div>\n</div>\n<div class="footer">\n<div class="footer-links">Copy
right &copy; LSE 2018   \n          <a href="https://www.lse.ac.uk/lse-
information/secure/report-a-page">Report a page</a> | <a href="http://www.lse.ac.uk/ab
outLSE/FOI/Home.aspx">Freedom of information</a> | <a href="http://www.lse.ac.uk/about
ThisWebsite/Home.aspx">About this site</a> | <a href="http://www.lse.ac.uk/newsAndMedi
a/lseinSocialMedia.aspx">Social media</a> <img src="http://www.lse.ac.uk/v4globa
l/images/SocialMediaFooter.jpg" alt="LSE Social media listings page" width="80" height
="15">\n</div>\n<div class="clear"> </div>\n</div>\n</div></body></html>\n'
```

Sending request in Python (continue)

Example of providing a wrong url:

In [8]:
```
url = 'https://www.lse.ac.uk/resources/calendar2021-2022/courseGuides/ST/2021_ST120.htm'
r = requests.get(url)
r.status_code, r.reason
```

Out[8]:

```
(404, 'Not Found')
```

In [9]:
```
r.text
```

Out[9]:

```
'<html>\r\n<head><title>404 Not Found</title></head>\r\n<body bgcolor="white">\r\n<cen
ter><h1>404 Not Found</h1></center>\r\n<hr><center>nginx/1.10.3 (Ubuntu)</center>\r\n
</body>\r\n</html>\r\n'
```

Collecting information on/from the Internet

- API (Application Programming Interface):
    - Set of defined rules that explains how computers or applications communicate with one another
- RSS (Rich Site Summary or Really Simple Syndication):
    - Provide subscribers with summary of the new content from frequently updated websites in standard format
        - Examples: news-related sites, blogs
- Web scraping:
    - Use software, scripts or by-hand extracting data from a web page

API demo

Get the search results related to "London School of Economics" on Wikipedia via an API call with the url **http://en.wikipedia.org/w/api.php?action=query&list=search&srsearch=London%20School%20of%20Economics&srlimit=1&format=js**

In [10]:

```
import requests
url = 'http://en.wikipedia.org/w/api.php?action=query&list=search&' \
    'srsearch=London%20School%20of%20Economics&srlimit=1&format=json'
r = requests.get(url)
r.text
```

Out[10]:


```
'{"batchcomplete":"","continue":{"sroffset":1,"continue":"-||"},"query":{"searchinfo":
{"totalhits":22118},"search":[{"ns":0,"title":"London School of Economics","pageid":67
704,"size":148603,"wordcount":13857,"snippet":"The <span class=\\"searchmatch\\">Londo
n</span> <span class=\\"searchmatch\\">School</span> <span class=\\"searchmatch\\">of
</span> <span class=\\"searchmatch\\">Economics</span> and Political Science (LSE) is
a public research university located in <span class=\\"searchmatch\\">London</span>, E
ngland, and a constituent college <span class=\\"searchmatch\\">of</span> the","timest
amp":"2023-02-27T11:31:49Z"}]}}'
```

API demo (continue)

In [11]:

```
import json

parsed = json.loads(r.text)
parsed
```

Out[11]:


```
{'batchcomplete': '',
 'continue': {'sroffset': 1, 'continue': '-||'},
 'query': {'searchinfo': {'totalhits': 22118},
  'search': [{'ns': 0,
    'title': 'London School of Economics',
    'pageid': 67704,
    'size': 148603,
    'wordcount': 13857,
    'snippet': 'The <span class="searchmatch">London</span> <span class="searchmatch">
School</span> <span class="searchmatch">of</span> <span class="searchmatch">Economics
</span> and Political Science (LSE) is a public research university located in <span c
lass="searchmatch">London</span>, England, and a constituent college <span class="sear
chmatch">of</span> the',
    'timestamp': '2023-02-27T11:31:49Z'}]}}
```


In [12]:

```
parsed['query']['search'][0]['snippet']
```

Out[12]:

```
'The <span class="searchmatch">London</span> <span class="searchmatch">School</span> <
span class="searchmatch">of</span> <span class="searchmatch">Economics</span> and Poli
tical Science (LSE) is a public research university located in <span class="searchmatc
h">London</span>, England, and a constituent college <span class="searchmatch">of</spa
n> the'
```

See **here** to learn more about Wikipedia API for searching.

RSS demo

Get the latest news from **BBC via RSS**:

In [15]:

```
import feedparser
url = 'http://feeds.bbci.co.uk/news/rss.xml'
feed = feedparser.parse(url)
feed.entries[0]
```

Out[15]:


```
{'title': 'Rishi Sunak vows to end asylum claims from small boat arrivals',
 'title_detail': {'type': 'text/plain',
  'language': None,
  'base': 'http://feeds.bbci.co.uk/news/rss.xml',
  'value': 'Rishi Sunak vows to end asylum claims from small boat arrivals'},
 'summary': 'Anyone coming to the UK on a small boat will be removed to Rwanda or anot
her country, under new laws.',
 'summary_detail': {'type': 'text/html',
  'language': None,
  'base': 'http://feeds.bbci.co.uk/news/rss.xml',
  'value': 'Anyone coming to the UK on a small boat will be removed to Rwanda or anoth
er country, under new laws.'},
 'links': [{'rel': 'alternate',
   'type': 'text/html',
   'href': 'https://www.bbc.co.uk/news/uk-64848101?at_medium=RSS&at_campaign=KARANG
A'}],
 'link': 'https://www.bbc.co.uk/news/uk-64848101?at_medium=RSS&at_campaign=KARANGA',
 'id': 'https://www.bbc.co.uk/news/uk-64848101',
 'guidislink': False,
 'published': 'Sun, 05 Mar 2023 15:34:30 GMT',
 'published_parsed': time.struct_time(tm_year=2023, tm_mon=3, tm_mday=5, tm_hour=15, t
m_min=34, tm_sec=30, tm_wday=6, tm_yday=64, tm_isdst=0)}
```

Web scraping demo

Get course title and essential reading list from **https://www.lse.ac.uk/resources/calendar2021-2022/courseGuides/ST/2021_ST115.htm**:

In [16]:
```
from bs4 import BeautifulSoup

url = 'https://www.lse.ac.uk/resources/calendar2022-2023/courseGuides/ST/2022_ST115.htm'
r = requests.get(url)
soup = BeautifulSoup(r.content,'lxml')
# get course title
print(soup.find('span', {'id': 'title'}).text.strip())
# get reading list
print(soup.find(text='Essential Reading:').find_next('ol').text)
```

```
Managing and Visualising Data

W. Mckinney, Python for Data Analysis, 2nd Edition, O'Reilly 2017
A. C. Muller and S. Guido, Introduction to Machine Learning with Python, O'Reilly, 201
6
Easley, David, and Jon Kleinberg. Networks, crowds, and markets: Reasoning about a hig
hly connected world. Cambridge university press, 2010
R. Ramakrishnan and J. Gehrke, Database Management Systems, 3rd Edition, McGraw Hill,
2002
```

HTML: A short tutorial

HTML

HTML (HyperText Markup Language) is a markup language that web pages are written in.

- HTML contains:
    - Content
    - *Tags* and *attributes* to describe how the browser should display the pages

HTML example

In [17]:

```
from IPython.display import HTML
html_code = """
<!DOCTYPE html>
<html>
    <head>
        <title>This is a title</title>
    </head>
    <body>
        <h2 style="color:red;">This is a header</h2>
        <p>This is a paragraph</p>
        <img src="figs/lse_logo.gif"/>
    </body>
</html>
"""
HTML(html_code)
```

Out[17]:

This is a header

This is a paragraph

HTML example (continue)

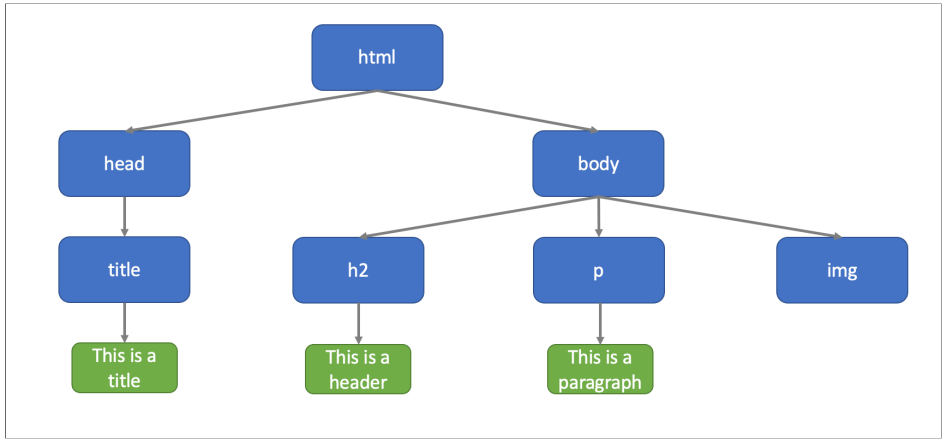
- `<!DOCTYPE html>`: a declaration that the page follows the HTML standard
- Pairs of tags like `<html>` and `</html>`
    - The first is a start or open tag, and the second is a close tag
- Some start tag like `h2` has some attributes like `style="color:red;"` which specifies the colour of the text

HTML elements and tags

HTML consists of HTML elements, *often* an HTML element starts with a start tag and ends with an end tag.

- Tags are written in angle brackets, *often* in pairs
- Tags are predefined. Tags used in the previous example:
    - `<html>` : Enclosed code is interpreted as HTML
    - `<head>` and `<body>` : Browser tab and browser content areas
    - `<title>` : Define the text that appears within the tab or window heading area
    - `<h2>` : Heading styles
    - `<p>` : A paragraph separated by lines
    - `<img>` : Self enclosing image tag with attributes

HTML: tree structure



By the way, this page is written in html (double click on the jupyter notebook version to reveal it).

HTML tags related to tables

HTML tag	use
<table>	Define a table
<tr>	Define a row of cells in a table
<th>	Define a cell in a table header row
<td>	Define a cell in a row

<thead> and <tbody> can be used as well to group header and body content.
(Double click on the jupyter notebook version to see the corresponding html code)

HTML tags related to list

- For unordered list
    - : Define an unordered list
    - : Define a list item
- For order list:
    1. : Define an ordered list
    2. : Define a list item (same as above)

(Double click on the jupyter notebook version to see the corresponding html code)

Some other example tags

- `<h1></h1> ... <h6></h6>` : Headings, with h1 largest and h6 smallest
- `<div></div>` : Define a section
- `<span></span>` : Define a (small) section
- `<a></a>` : link tag (e.g. hyperlink in the text) with attributes

Example use of tags

In [18]:

```
html_code = """
<!DOCTYPE html>
<html>
    <h3>This is an h3 header</h3>
    <div style="color:blue;">
        <p>This is a simple section with a link and an image:</p>
        <center>
            <a href='https://www.lse.ac.uk/resources/calendar2021-2022/courseGuides/ST/2021_ST101.htm'>
                example link
            </a>
            <br>
            <img src='https://www.lse.ac.uk/Statistics/Assets/Images/General/Globe1.jpg' width=150/>
        </center>
        Note all text has the same style in the same div.
    </div>
</html>"""
HTML(html_code)
```

Out[18]:

This is an h3 header

This is a simple section with a link and an image:

example link

Note all text has the same style in the same div.

Web scraping

Web scraping

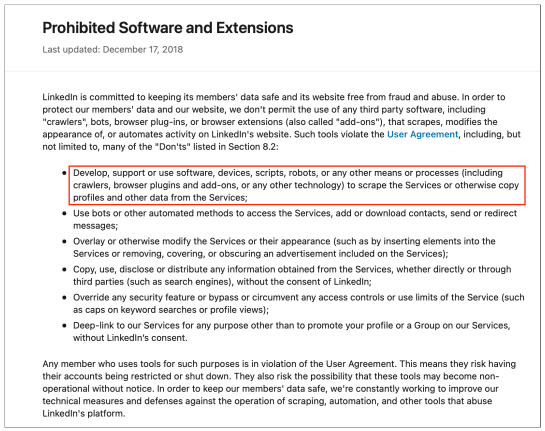
Web scraping (or web harvesting, or web data extraction) is to extract data from websites.

- While web scraping can be done manually, the term typically refers to automated processes implemented using a bot or web crawler
- Advantages:
    - Vast source of updated information that may not be available in other ways
    - Automate data downloading tasks

Warning

Web scraping is often not welcomed by website owners. Beware of computational and legal issues related with web scrapping

- Read the terms of use of a website. Most sites prohibit you from using the data for commercial purposes
    - Some prohibit you doing it for any purpose (especially those who provide *paid service*):



Prohibited Software and Extensions

Last updated: December 17, 2018

LinkedIn is committed to keeping its members' data safe and its website free from fraud and abuse. In order to protect our members' data and our website, we don't permit the use of any third party software, including "crawlers", bots, browser plug-ins, or browser extensions (also called "add-ons"), that scrapes, modifies the appearance of, or automates activity on LinkedIn's website. Such tools violate the User Agreement, including, but not limited to, many of the "Don'ts" listed in Section 8.2:

- Develop, support or use software, devices, scripts, robots, or any other means or processes (including crawlers, browser plugins and add-ons, or any other technology) to scrape the Services or otherwise copy profiles and other data from the Services;
- Use bots or other automated methods to access the Services, add or download contacts, send or redirect messages;
- Overlay or otherwise modify the Services or their appearance (such as by inserting elements into the Services or removing, covering, or obscuring an advertisement included on the Services);
- Copy, use, disclose or distribute any information obtained from the Services, whether directly or through third parties (such as search engines), without the consent of LinkedIn;
- Override any security feature or bypass or circumvent any access controls or use limits of the Service (such as caps on keyword searches or profile views);
- Deep-link to our Services for any purpose other than to promote your profile or a Group on our Services, without LinkedIn's consent.

Any member who uses tools for such purposes is in violation of the User Agreement. This means they risk having their accounts being restricted or shut down. They also risk the possibility that these tools may become non-operational without notice. In order to keep our members' data safe, we're constantly working to improve our technical measures and defenses against the operation of scraping, automation, and other tools that abuse LinkedIn's platform.

- Read the *robots exclusion protocol*

Robots exclusion protocol

Robots exclusion protocol specifies which areas of the website should not be processed or scanned by robots.

- The robots exclusion protocol is located at the top-level directory of the web server. You can by adding `/robots.txt` to a website's URL
    - Example: **https://en.wikipedia.org/robots.txt**
- Purpose of robots exclusion protocol:
    - Hide things from robots
    - Improve search engine optimization by excluding duplicate content

Read the terms of use if you are not sure.

Warning (continue)

- Be polite and careful
    - No spam or overloading the websites. You may be blocked if you send out excessive requests
    - Scrape in off-peak times
- Ethical issues? Privacy concerns?
    - Scraping the Personal Identifiable Information of EU citizens may breach General Data Protection Regulation (GDPR)
- If possible, use API instead

HTML and web scraping

Languages used in creating web pages:

- HTML: the basic structure of web pages, display mostly static contents
- CSS: Improve the aesthetics of web pages by providing more control in terms of presentation, formatting, layout, etc
- JavaScript: Control dynamic behavior and allow interactive web sites
    - Many contents of dynamic web pages (e.g. Google Map) cannot be found anywhere in html

In this course we only learn about scraping content written in HTML.

Web-scraping steps

1. Get the contents from the web
    - `requests` : simple HTTP library to send requests to servers
2. Extract information:
    - `BeautifulSoup` : parse HTML (and XML) and help to navigate, search and extract the data from the parsed data
3. Reshape and save the information as data

Web scraping example 1: **example.com**

Step 1: Get contents from the website using `requests`:

In [19]:
```
url = 'http://example.com'
r = requests.get(url)
print(r.content)
```

```
b'<!doctype html>\n<html>\n<head>\n    <title>Example Domain</title>\n\n    <meta char
set="utf-8" />\n    <meta http-equiv="Content-type" content="text/html; charset=utf-8"
/>\n    <meta name="viewport" content="width=device-width, initial-scale=1" />\n    <s
tyle type="text/css">\n    body {\n        background-color: #f0f0f2;\n        margin:
0;\n        padding: 0;\n        font-family: -apple-system, system-ui, BlinkMacSystem
Font, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;\n
\n    }\n    div {\n        width: 600px;\n        margin: 5em auto;\n        padding:
2em;\n        background-color: #fdfdff;\n        border-radius: 0.5em;\n        box-s
hadow: 2px 3px 7px 2px rgba(0,0,0,0.02);\n    }\n    a:link, a:visited {\n        colo
r: #38488f;\n        text-decoration: none;\n    }\n    @media (max-width: 700px) {\n
div {\n            margin: 0 auto;\n            width: auto;\n        }\n    }\n    </
style>    \n</head>\n\n<body>\n<div>\n    <h1>Example Domain</h1>\n    <p>This domain
is for use in illustrative examples in documents. You may use this\n    domain in lite
rature without prior coordination or asking for permission.</p>\n    <p><a href="http
s://www.iana.org/domains/example">More information...</a></p>\n</div>\n</body>\n</html
>\n'
```

Example 1: parse content by BeautifulSoup

Step 2: Extract information from html using `BeautifulSoup`:

In [20]:
```
# you may need to install the module before using it
from bs4 import BeautifulSoup
soup = BeautifulSoup(r.content,'lxml')
print(type(soup))
```

```
<class 'bs4.BeautifulSoup'>
```

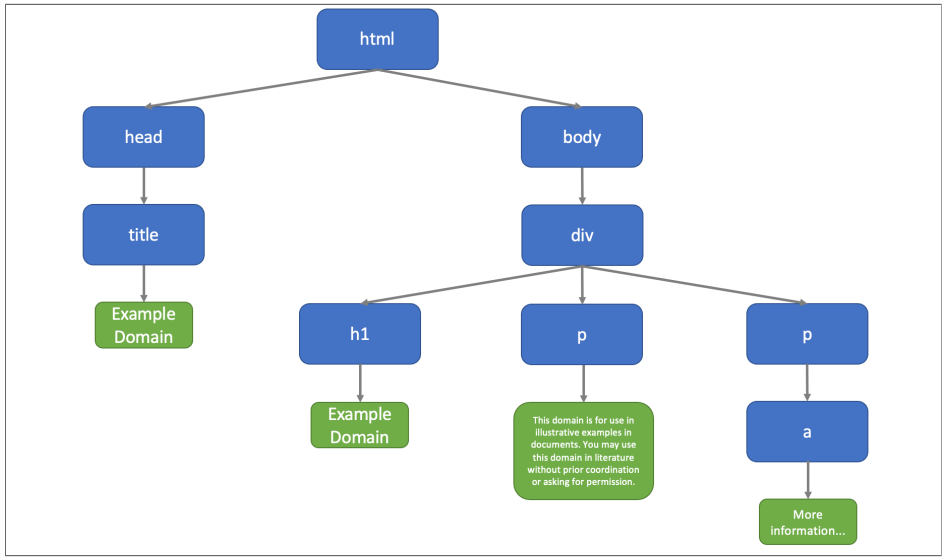
In [21]:
```
soup.contents
```

Out[21]:

```
['html',
 <html>
 <head>
 <title>Example Domain</title>
 <meta charset="utf-8"/>
 <meta content="text/html; charset=utf-8" http-equiv="Content-type"/>
 <meta content="width=device-width, initial-scale=1" name="viewport"/>
 <style type="text/css">
     body {
         background-color: #f0f0f2;
         margin: 0;
         padding: 0;
         font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open
Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

     }
     div {
         width: 600px;
         margin: 5em auto;
```

```
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>
<body>
<div>
<h1>Example Domain</h1>
<p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
<p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>,
'\n']
```

Example 1: Visualise the tree structure



Get an element by its tag

Find the *first* appearance of the element with the given tag using `find()`. For example:

In [22]:
```
section = soup.find('div')
section
```

Out[22]:

```
<div>
<h1>Example Domain</h1>
<p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
<p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
```

Note you can also do the following:

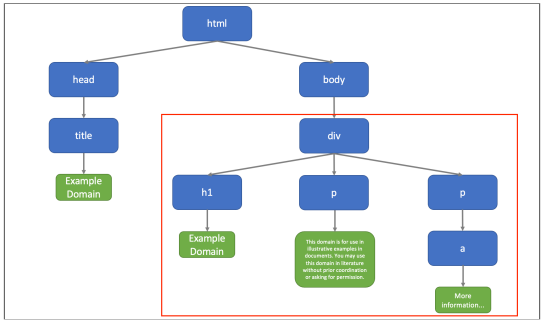
In [23]:
```
soup.div
```

Out[23]:

```
<div>
<h1>Example Domain</h1>
<p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
<p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
```

Get an element by its tag (continue)

- Inside `soup.find('div')` there are some other elements with tags `<h1>`, `<p>`



- These elements are their children and we can continue to navigate (recursively) to other elements. For example:

In [24]:
```
section.find('h1')
```

Out[24]:

```
<h1>Example Domain</h1>
```

To get the text, use `.text`:

In [25]:
```
section.find('h1').text
```

Out[25]:

```
'Example Domain'
```

Get all elements with the given tag

Note with `find()` we only get the first element with the given tag. For example:

In [26]:
```
soup.find('p')
```

Out[26]:

```
<p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
```

If we want to get all the elements, use `find_all()` instead. Example:

In [27]:
```
soup.find_all('p') # return a list
```

Out[27]:

```
[<p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>,
 <p><a href="https://www.iana.org/domains/example">More information...</a></p>]
```

Get all elements with the given list of tags

Apart from providing a tag, you can also provide a list of tags:

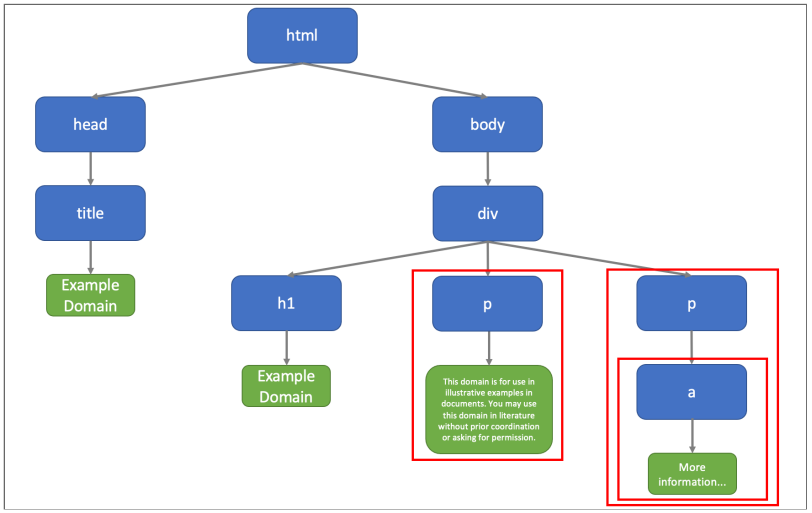
In [28]:
```
soup.find_all(['p', 'a'])
```

Out[28]:

```
[<p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>,
 <p><a href="https://www.iana.org/domains/example">More information...</a></p>,
 <a href="https://www.iana.org/domains/example">More information...</a>]
```

Note that `<a href="https://www.iana.org/domains/example">More information...</a>` appear twice! Why?

Graphical explanation



`find_all()` by default the search is done recursively. So as `find()`.

Get elements with the given tag: some more examples

`None` is returned when the tag cannot be found for `find()`, and empty list for `find_all()`. For example:

In [29]:
```
print(soup.find('img'))
```

```
None
```

In [30]:
```
print(soup.find_all('img'))
```

```
[]
```

As `find_all()` returns a list, we can iterate over each of the element within it. For example:

In [31]:
```
for paragraph in soup.find_all('p'):
    print(paragraph.text)
    print('------')
```

```
This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.
------
More information...
------
```

Get attributes

Some elements have some attributes, for example `href="https://..."` is the attribute of the following element:

In [32]:
```
soup.find('a')
```

Out[32]:

```
<a href="https://www.iana.org/domains/example">More information...</a>
```

You can extract the attribute by `['name of attribute']`. For example:

In [33]:
```
soup.find('a')['href']
```

Out[33]:

```
'https://www.iana.org/domains/example'
```

Get elements by text

If you want to match text instead of tag, you can use the argument `string`. For example:

In [34]:
```
elements = soup.find_all(string = 'Example Domain')
elements
```

Out[34]:

```
['Example Domain', 'Example Domain']
```

While it seems you just get back a list of the same "string", it is not the case:

In [35]:
```
type(elements[0])
```

Out[35]:

```
bs4.element.NavigableString
```

You can for example do something like:

In [36]:
```
elements[1].find_next('p') # demonstrate later
```

Out[36]:

```
<p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
```

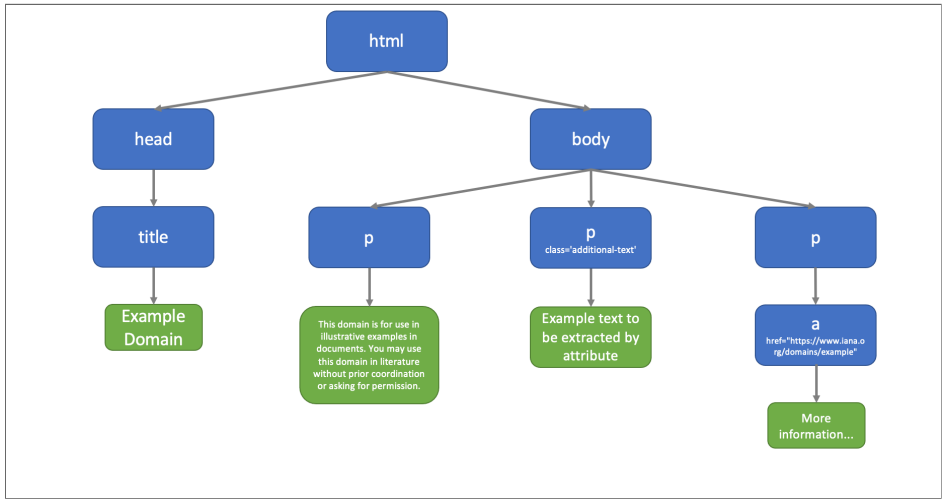
Modified example

Here we add a `<p>` with an *attribute*, for which it will be easier for us to extract the corresponding data later.

In [37]:

```
            html_code = """
<!DOCTYPE html>
<html>
    <head>
        <title>Example Domain</title>
    </head>
    <body>
        <p>This domain is for use in illustrative examples in documents. You may use this
            domain in literature without prior coordination or asking for permission.</p>
        <p class='additional-text'>Example text to be extracted by attribute</p>
        <p><a href="https://www.iana.org/domains/example">More information...</a></p>
    </body>
</html>"""
soup = BeautifulSoup(html_code,'lxml')
```

Modified example: tree



Search using attribute

For the find functions like `find()` and `find_all()`, you can provide the attribute to be matched.

- For example, here we get the first `<p>` with `class='additional-text'`:

In [38]:
```
p = soup.find('p', attrs={'class': 'additional-text'})
p
```

Out[38]:

```
<p class="additional-text">Example text to be extracted by attribute</p>
```

Compare to:

In [39]:
```
soup.find('p')
```

Out[39]:

```
<p>This domain is for use in illustrative examples in documents. You may use this
            domain in literature without prior coordination or asking for permission.
</p>
```

Search the tree

Use `find_previous()` and `find_next()` if you want to find the tag before and after the current location of the document. Example:

In [40]:
```
p.find_previous('p') # previous <p>
```

Out[40]:

```
<p>This domain is for use in illustrative examples in documents. You may use this
            domain in literature without prior coordination or asking for permission.
</p>
```

In [41]:
```
p.find_next('a') # next <a>
```

Out[41]:

```
<a href="https://www.iana.org/domains/example">More information...</a>
```

The following does not work as the element with `<p class='additional-text'>` does not have any children with tag `<p>` :

In [42]:
```
p.find('p')
```

Search the tree (continue)

There is also `find_all_previous()` and `find_all_next()`. Example:

In [43]:
```
p.find_all_next(['p','a'])
```

Out[43]:

```
[<p><a href="https://www.iana.org/domains/example">More information...</a></p>,
 <a href="https://www.iana.org/domains/example">More information...</a>]
```

Web scraping example 2 (previous demo)

You should now understand the code below:

In [44]:

```
url = 'https://www.lse.ac.uk/resources/calendar2022-2023/courseGuides/ST/2022_ST115.htm'
r = requests.get(url)
soup = BeautifulSoup(r.content,'lxml')
# get course title
print(soup.find('span', {'id': 'title'}).text.strip())
# get reading list
print(soup.find(text='Essential Reading:').find_next('ol').text)
```

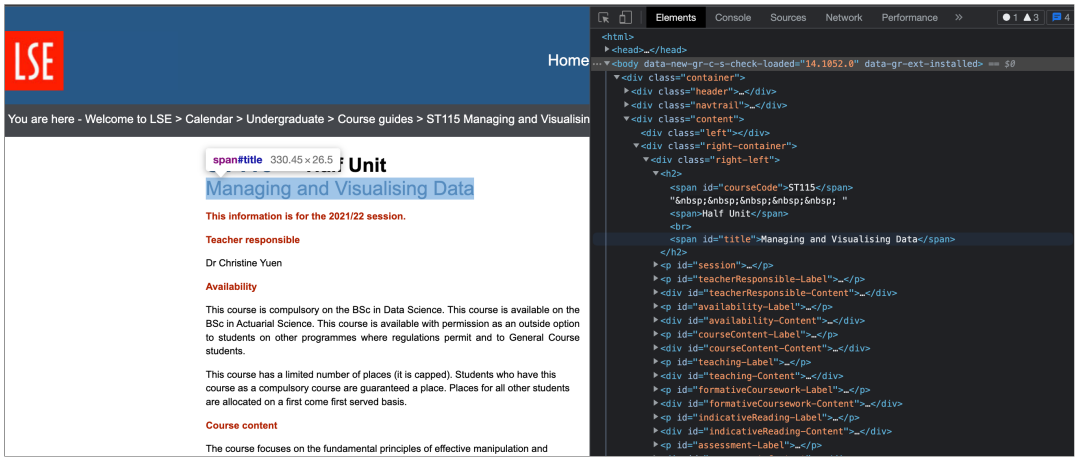
```
Managing and Visualising Data

W. Mckinney, Python for Data Analysis, 2nd Edition, O'Reilly 2017
A. C. Muller and S. Guido, Introduction to Machine Learning with Python, O'Reilly, 201
6
Easley, David, and Jon Kleinberg. Networks, crowds, and markets: Reasoning about a hig
hly connected world. Cambridge university press, 2010
R. Ramakrishnan and J. Gehrke, Database Management Systems, 3rd Edition, McGraw Hill,
2002
```

But how can you know which tags, attributes, etc to use, especially when the html code can be very long?

Developer tools

To inspect a specific web element in Chrome, select and right-click on the element you want to extract the information and select the Inspect option

- Example:



Web scraping example 3: Extracting table

You should now understand the following code, and also able to write the following code:

In [45]:
```
import pandas as pd
url = 'http://www.worldometers.info/world-population/population-by-country/'
page = requests.get(url)
soup = BeautifulSoup(page.content,'lxml')
result =[]
for row in soup.find('table').find('tbody').find_all('tr'):
    result.append([cell.text for cell in row.find_all('td')])
pd.DataFrame(result, columns=[cell.text for cell in soup.find('thead').find('tr').find_all('th')])
```

Out[45]:

	#	Country (or dependency)	Population (2020)	Yearly Change	Net Change	Density (P/Km²)	Land Area (Km²)	Migrants (net)	Fert. Rate	Med. Age	Urban Pop %	World S
0	1	China	1,439,323,776	0.39 %	5,540,090	153	9,388,211	-348,399	1.7	38	61 %	18.4
1	2	India	1,380,004,385	0.99 %	13,586,631	464	2,973,190	-532,687	2.2	28	35 %	17.7
2	3	United States	331,002,651	0.59 %	1,937,734	36	9,147,420	954,806	1.8	38	83 %	4.2
3	4	Indonesia	273,523,615	1.07 %	2,898,047	151	1,811,570	-98,955	2.3	30	56 %	3.5
4	5	Pakistan	220,892,340	2.00 %	4,327,022	287	770,880	-233,379	3.6	23	35 %	2.8
...	...	...	...	...	...	...	...	...	...	...	...	
230	231	Montserrat	4,992	0.06 %	3	50	100		N.A.	N.A.	10 %	0.0
231	232	Falkland Islands	3,480	3.05 %	103	0	12,170		N.A.	N.A.	66 %	0.0
232	233	Niue	1,626	0.68	11	6	260		N.A.	N.A.	46	0.0

	#	Country (or dependency)	Population (2020)	Yearly Change	Net Change	Density (P/Km²)	Land Area (Km²)	Migrants (net)	Fert. Rate	Med. Age	Urban Pop %	World S
				%							%	
233	234	Tokelau	1,357	1.27 %	17	136	10		N.A.	N.A.	0 %	0.0
234	235	Holy See	801	0.25 %	2	2,003	0		N.A.	N.A.	N.A.	0.0

235 rows × 12 columns

But there is a much easier way to scrape the table withe the use of pandas!

Example 3: using pandas

You can easily scrap the table with the use of `pd.read_html()`:

```
In [46]:
pd.read_html(page.content)[0]
```

Out[46]:

	#	Country (or dependency)	Population (2020)	Yearly Change	Net Change	Density (P/Km²)	Land Area (Km²)	Migrants (net)	Fert. Rate	Med. Age	Urban Pop %	World Share
0	1	China	1439323776	0.39 %	5540090	153	9388211	-348399.0	1.7	38	61 %	18.47 %
1	2	India	1380004385	0.99 %	13586631	464	2973190	-532687.0	2.2	28	35 %	17.70 %
2	3	United States	331002651	0.59 %	1937734	36	9147420	954806.0	1.8	38	83 %	4.25 %
3	4	Indonesia	273523615	1.07 %	2898047	151	1811570	-98955.0	2.3	30	56 %	3.51 %
4	5	Pakistan	220892340	2.00 %	4327022	287	770880	-233379.0	3.6	23	35 %	2.83 %
...	...	...	...	...	...	...	...	...	...	...	...	...
230	231	Montserrat	4992	0.06 %	3	50	100	NaN	N.A.	N.A.	10 %	0.00 %
231	232	Falkland Islands	3480	3.05 %	103	0	12170	NaN	N.A.	N.A.	66 %	0.00 %
232	233	Niue	1626	0.68 %	11	6	260	NaN	N.A.	N.A.	46 %	0.00 %
233	234	Tokelau	1357	1.27 %	17	136	10	NaN	N.A.	N.A.	0 %	0.00 %

	#	Country (or dependency)	Population (2020)	Yearly Change	Net Change	Density (P/Km²)	Land Area (Km²)	Migrants (net)	Fert. Rate	Med. Age	Urban Pop %	World Share
234	235	Holy See	801	0.25 %	2	2003	0	NaN	N.A.	N.A.	N.A.	0.00 %

235 rows × 12 columns

Exercise

Get the list of all LSE directors from the wikipedia page.

In [47]:
```
page = requests.get('https://en.wikipedia.org/wiki/London_School_of_Economics')
pd.read_html(page.content)[2]
```

Out[47]:

	Years	Name
0	1895–1903	William Hewins
1	1903–1908	Sir Halford Mackinder
2	1908–1919	The Hon. William Pember Reeves
3	1919–1937	Lord Beveridge
4	1937–1957	Sir Alexander Carr-Saunders
5	1957–1967	Sir Sydney Caine
6	1967–1974	Sir Walter Adams
7	1974–1984	Lord Dahrendorf
8	1984–1990	Indraprasad Gordhanbhai Patel
9	1990–1996	Sir John Ashworth
10	1996–2003	Lord Giddens
11	2003–2011	Sir Howard Davies
12	2011–2012	Dame Judith Rees
13	2012–2016†	Craig Calhoun
14	2016–2017	Julia Black

	Years	Name
15	2017–present	Dame Nemat Shafik (1st President and VC of LSE)

Important practical tips on web scraping

- Do not request too many times in a short period of time
    - You can use `time.sleep()` and `random()` to prevent sending too many requests in a short period of time, and make it look "less like a bot". Example:

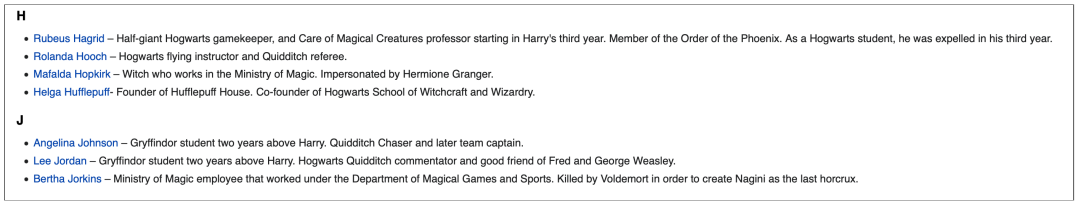
In [48]:
```
import time
import random
for course_code in ['ST101', 'ST115', 'ST207']:
    url = f'https://www.lse.ac.uk/resources/calendar2021-2022/courseGuides/ST/'\
        f'2021_{course_code}.htm'
    r = requests.get(url)
    soup = BeautifulSoup(r.content,'lxml')
    print(course_code, soup.find('span', {'id': 'title'}).text.strip())
    time.sleep(random.randint(1,5))
```

```
ST101 Programming for Data Science
ST115 Managing and Visualising Data
ST207 Databases
```

- Web scraping can be time consuming - spare enough time to collect the data

Important practical tips on web scraping (continue)

- Web pages are not written with the intention for you to collect the data
    - There is no guarantee structure of web pages
        - For example, for the list of **Harry Potters characters on wikipedia page**, a quick glance of the list seems to suggest that the data is in the format of `[name] — [descriptive]` but there are some exceptions:



H

- Rubeus Hagrid – Half-giant Hogwarts gamekeeper, and Care of Magical Creatures professor starting in Harry's third year. Member of the Order of the Phoenix. As a Hogwarts student, he was expelled in his third year.
- Rolanda Hooch – Hogwarts flying instructor and Quidditch referee.
- Mafalda Hopkirk – Witch who works in the Ministry of Magic. Impersonated by Hermione Granger.
- Helga Hufflepuff- Founder of Hufflepuff House. Co-founder of Hogwarts School of Witchcraft and Wizardry.

J

- Angelina Johnson – Gryffindor student two years above Harry. Quidditch Chaser and later team captain.
- Lee Jordan – Gryffindor student two years above Harry. Hogwarts Quidditch commentator and good friend of Fred and George Weasley.
- Bertha Jorkins – Ministry of Magic employee that worked under the Department of Magical Games and Sports. Killed by Voldemort in order to create Nagini as the last horcrux.

    - The structure of a web page can change as they like. For example, there is no guarantee that the title always have the attribute `id="title"`. You may need to rewrite your code if they decide to change their html code
- Use API if possible

API

API

API (Application programming interface) is a set of rules and specifications that software programs can follow to communicate with each other.

- It serves as an interface between different software programs and facilitates their interaction
- How an API works:
    - A client application initiates an API call to request information
        - Request includes: request method, headers. etc
    - If the request is valid, the API makes a call to the external program or web server
    - The server sends a response to the API with the requested information
    - The API transfers the data to the initial requesting application

Sound familiar?

API vs web scraping

- Advantage of API Scraping:
    - API calls are faster than web scraping
    - Data from API is in more standard format
    - Authorised way to access data
- Limitations of API Scraping:
    - Availability: not all websites have APIs
    - Most APIs have limited usage policy
        - Rate limit: number of API calls, a user can make within a given time period
        - Can be different for different types of users (e.g. free and paid users)
            - e.g. For normal developer account, users can only get back up to previous 7 day tweets

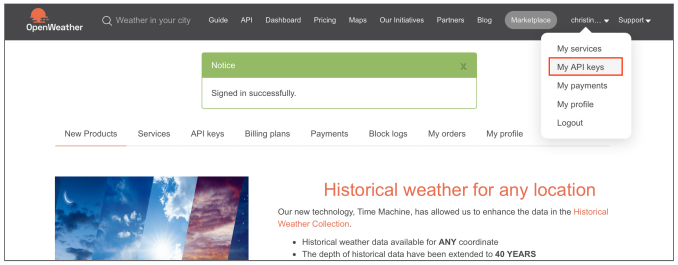
Example 1: OpenWeather API

OpenWeather provides API for users to retrieve weather data, including real time and historical weather.

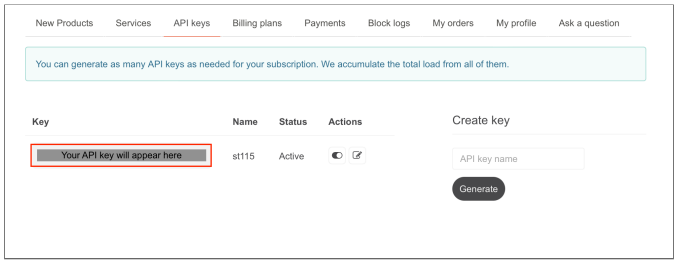
- See the whole list of data provided **here**

How to use OpenWeather API

- Before using it, please first open an account **here**
- Then at the top right, click your account -> My API keys:



- Copy your API key from the page:



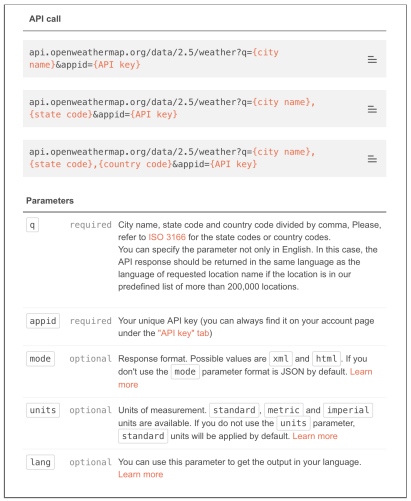
API Keys

To prevent unauthorised use of API, API calls are restricted to those that provide proper authentication credentials, and these credentials are in the form of an *API key*.

- API key: a unique identifier that authenticates and associates with a user or an app
    - For now, you can consider API key is your log in details. Just like your other log in details, you should NOT share and expose it to others!

OpenWeather API: Get the current weather information

Follow the instruction from the **official document**:



API call

```
api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}
```

```
api.openweathermap.org/data/2.5/weather?q={city name},{state code}&appid={API key}
```

```
api.openweathermap.org/data/2.5/weather?q={city name},{state code},{country code}&appid={API key}
```

Parameters

`q`	required	City name, state code and country code divided by comma, Please, refer to ISO 3166 for the state codes or country codes. You can specify the parameter not only in English. In this case, the API response should be returned in the same language as the language of requested location name if the location is in our predefined list of more than 200,000 locations.
`appid`	required	Your unique API key (you can always find it on your account page under the "API key" tab)
`mode`	optional	Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default. Learn more
`units`	optional	Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default. Learn more
`lang`	optional	You can use this parameter to get the output in your language. Learn more

Get the current weather information (continue)

Get the current weather of London using the given API call:

In [58]:

```
            # I stored my key here to hide it from you
with open('keys.json') as f:
    keys = json.load(f)
# But for now you can just change the following line and put your api key
# at the right hand side
api_key = keys['open_weather']['api_key']

city_name = 'London'
url = f'http://api.openweathermap.org/data/2.5/weather?q={city_name}&appid={api_key}'
r = requests.get(url)
r
```

Out[58]:

```
<Response [401]>
```

In [49]:

```
            r.text
```

Out[49]:


```
'{"coord":{"lon":-0.1257,"lat":51.5085},"weather":[{"id":804,"main":"Clouds","description":"overcast clouds","icon":"04d"}],"base":"stations","main":{"temp":282.72,"feels_like":281.42,"temp_min":280.64,"temp_max":284.46,"pressure":1018,"humidity":79},"visibility":10000,"wind":{"speed":2.57,"deg":140},"clouds":{"all":100},"dt":1646900275,"sys":{"type":2,"id":2019646,"country":"GB","sunrise":1646893583,"sunset":1646934929},"timezone":0,"id":2643743,"name":"London","cod":200}'
```

Get the current weather information (continue)

In [50]:
```
weather = json.loads(r.text)
```

In [51]:
```
weather['main']['temp']
```

Out[51]:

```
282.72
```

Why is the number so high?

Get the current weather information (continue)

Provide additional argument to have the right unit of weather details:

In [52]:
```
url = f'http://api.openweathermap.org/data/2.5/weather'\
    f'?q={city_name}&appid={api_key}&units=metric'
r = requests.get(url)
r
```

Out[52]:

```
<Response [200]>
```

In [53]:
```
weather = json.loads(r.text)
weather['main']['temp']
```

Out[53]:

```
9.56
```

Lesson: Always read the official documentation - do not assume it will do what you *think* it will do.

Recall: URL

- A typical URL could have the form: `<scheme>://<host>:<port>/<path><;parameters><?query><#fragment>`
    - Scheme (e.g., http, https)
    - Location of the server, e.g. a hostname or IP address
    - [optional] port: communication endpoint
    - Path to the resource on the server
    - [optional] parameters (;something), query string (?something), fragment identifier (#something)

More on API

- Providers often decides the API, and clients have to read the documentations to know how to use their API
- Today we have only provided a very brief introduction and simple demonstration of the use of API. We will continue our discussion and demostration of API in the workshop

Summary

Collecting data from the Internet.

- Web scraping
    - Examples
    - Things to be aware of when collecting data via web scraping

Preview of workshop

Get data using API.

Further readings

- Python for Data Analysis Ch 6.1 on XML and HTML
- **BeautifulSoup quick start**
- **Requests quick start**
- APIs and Web Scraping in Python from **DataQuest**