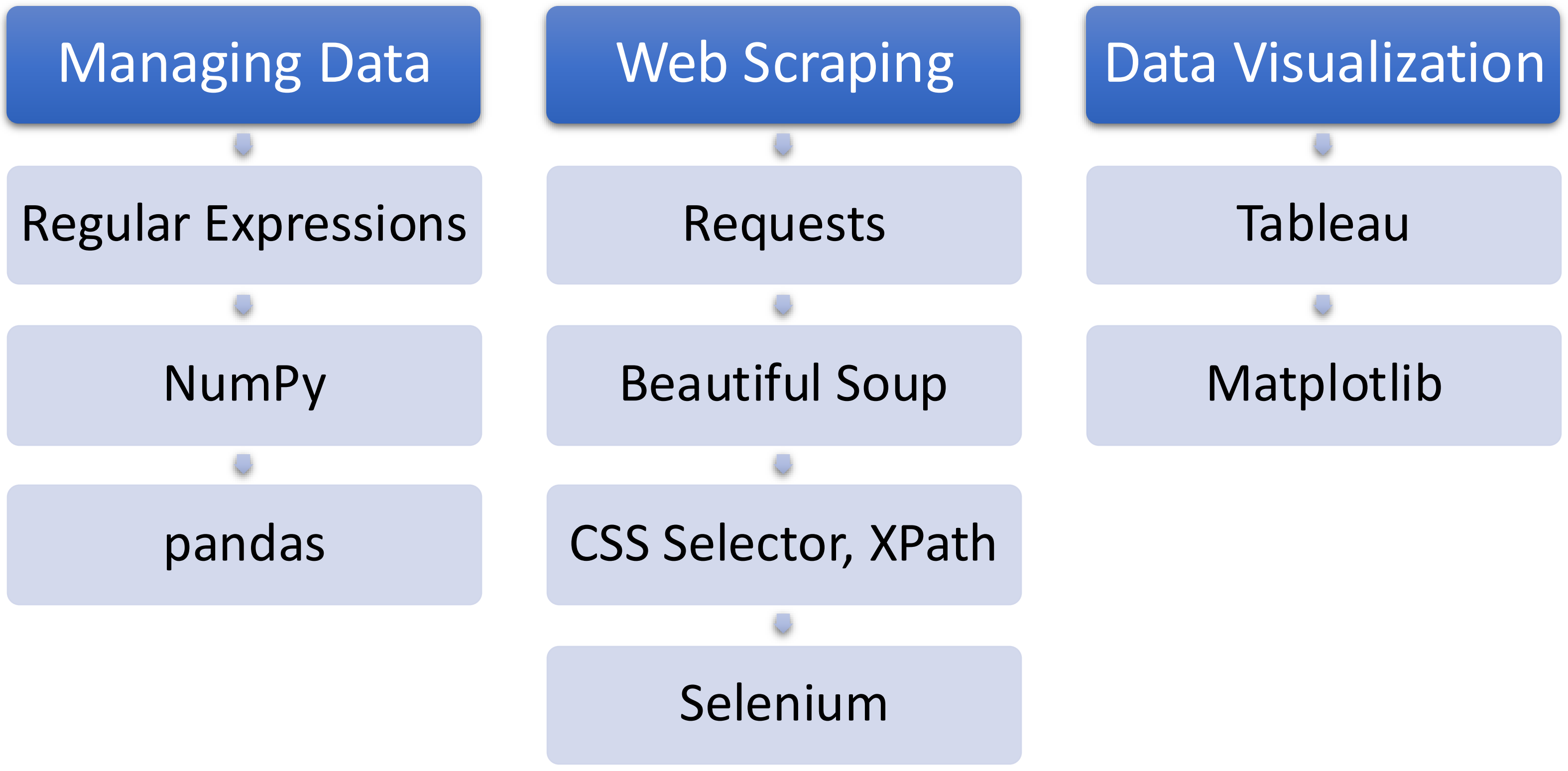
Web Scraping I

MSBA7001 Business Intelligence and Analytics
HKU Business School
The University of Hong Kong

Instructor: Dr. DING Chao

Course Roadmap



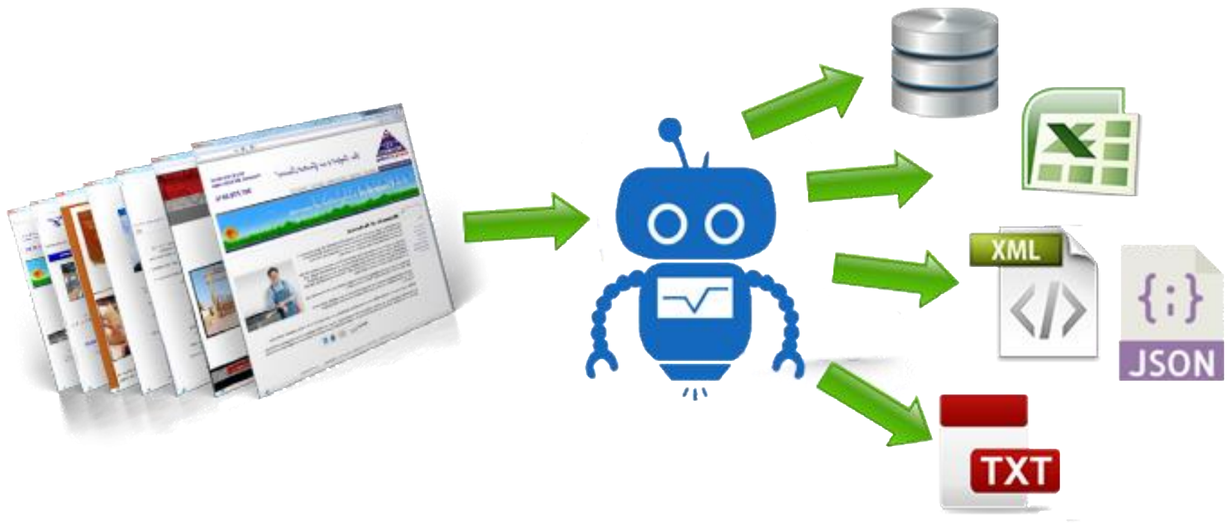
Agenda

- What is Web Scraping
- Reading Web Pages (the requests module)
- HTML Page
- BeautifulSoup 4

What is Web Scraping

What is Web Scraping?

- When a program or script pretends to be a browser and retrieves web pages, looks at those web pages, extracts information, and then looks at more web pages.
- Search engines like Google scrape web pages - we call this “spidering the web” or “web crawling”.

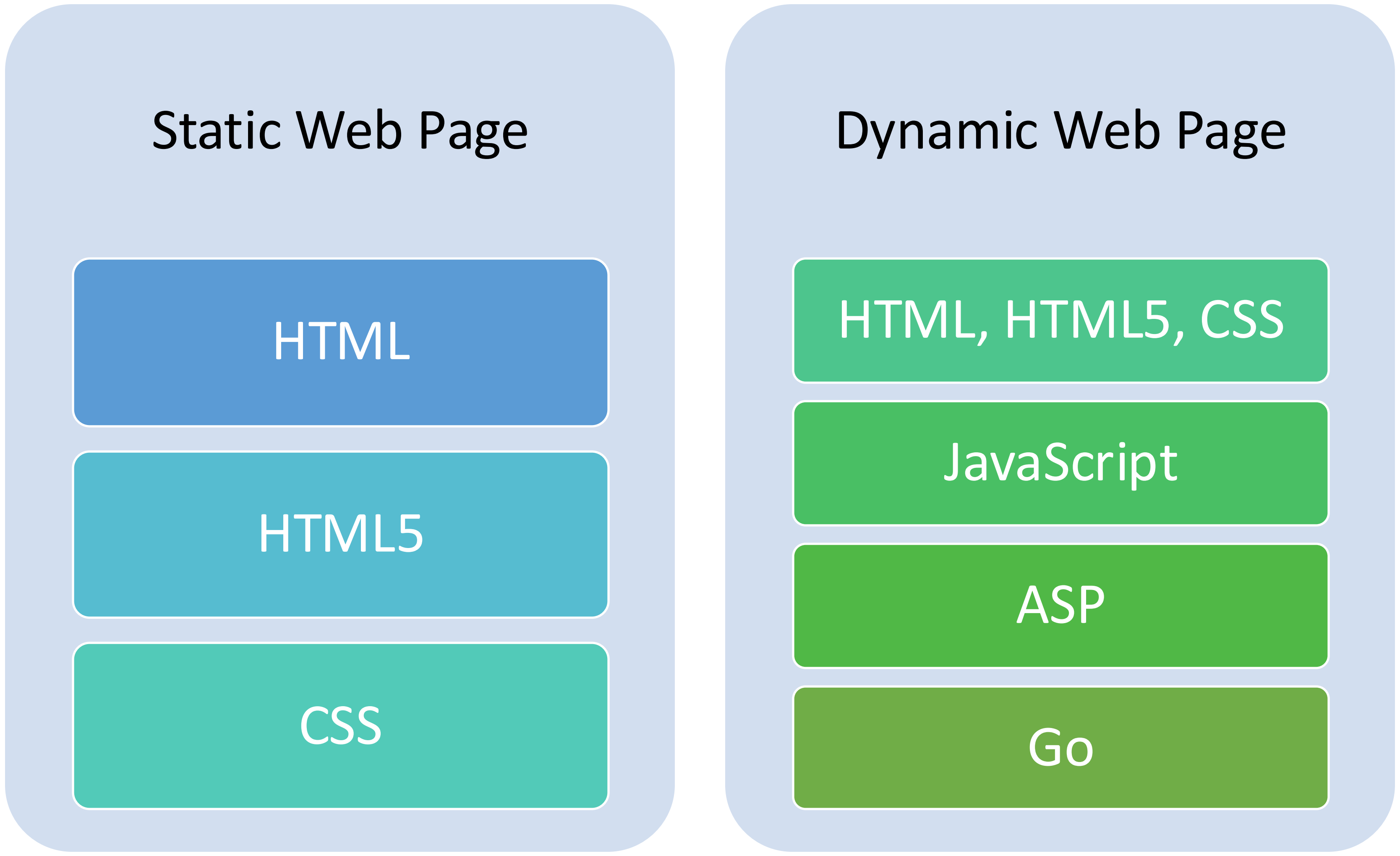


Why Scraping?

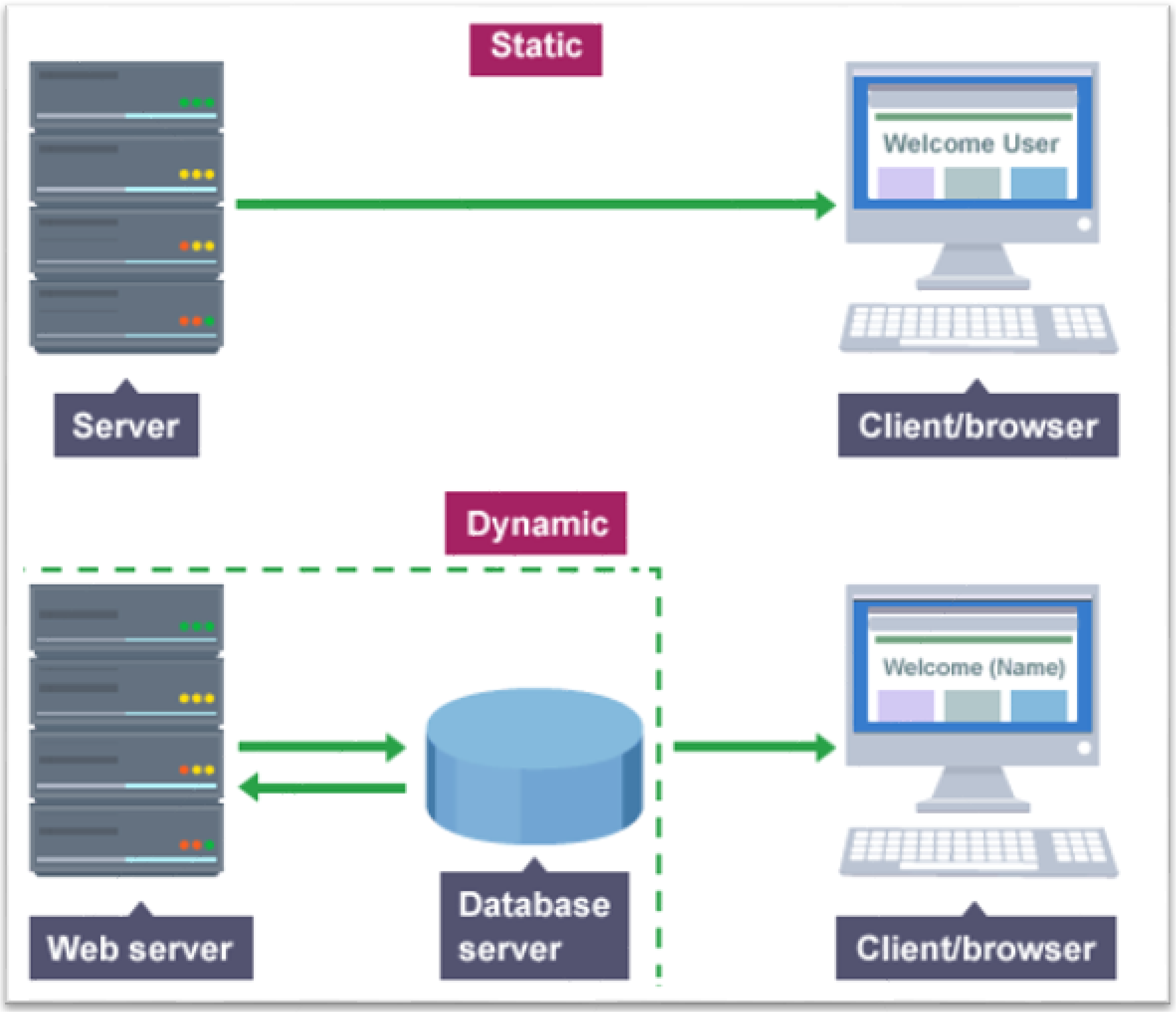
- Pull data for scientific research.
- Get your own data back out of some system that has no “export capability”.
- Monitor a site for new information (e.g., prices).
- Spider the web to make a database for a search engine.
- Websites now increasing implement anti-scraping techniques, making it more and more difficult to scrape the web.

Web Pages

- A very rough idea of how the world of web pages are created.

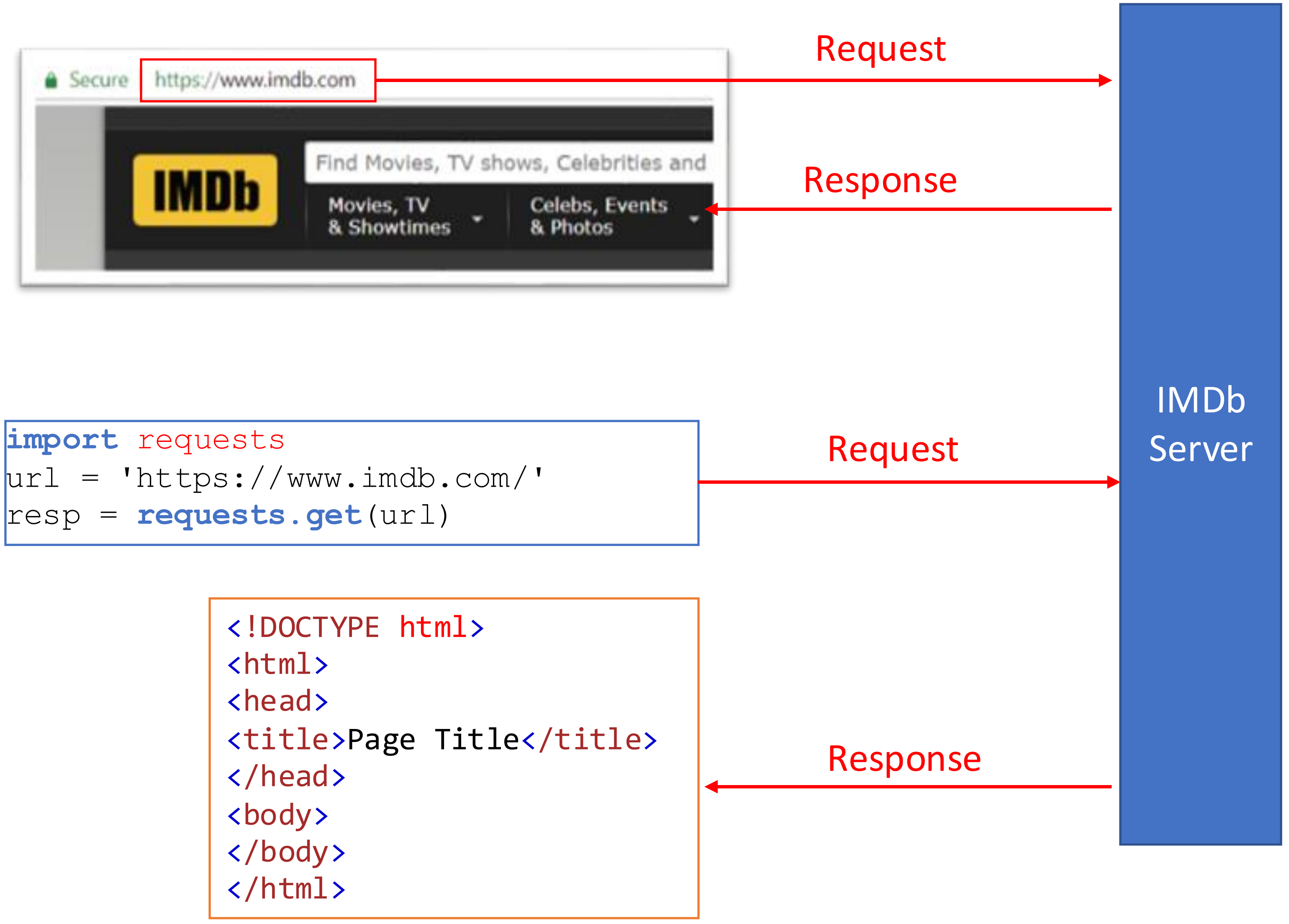


Web Pages


Static
Welcome User
Server
Client/browser
Dynamic
Welcome (Name)
Web server
Database server
Client/browser

Reading Web Pages

Request and Response



```python
import requests
url = 'https://www.imdb.com/'
resp = requests.get(url)
```

```html
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
</body>
</html>
```

The `requests` module

- The `requests` module allows you to send HTTP requests using Python.
- An HTTP request is meant to either retrieve data from a specified URL or to push data to a server.
- It works as a request-response protocol between a client and a server.
- For more details:

https://realpython.com/python-requests/

Methods

```
import requests
```

Method	Description
delete(url)	Sends a DELETE request to the specified url
get(url)	Sends a GET request to the specified url
head(url)	Sends a HEAD request to the specified url
patch(url, data)	Sends a PATCH request to the specified url
post(url, data)	Sends a POST request to the specified url
put(url, data)	Sends a PUT request to the specified url

The `get` method

- `get` method is used to retrieve information from the given server using a given URL. It returns a response object.
- Basic syntax:

```
requests.get(url, params={key: value}, args)
```

```
url = 'http://www.example.com'
resp = requests.get(url)
```

This is a response object

Attributes of Response Object

Attribute	Description
text	Returns the content of the response, in Unicode (string)
content	Returns the content of the response, in bytes
headers	Returns a dictionary of response headers
url	Returns the URL of the response
status_code	Returns a number that indicates the status
ok	Returns True if status_code is less than 400, otherwise False

```
print(resp.url)
```

http://www.example.com/

Status Code

- HTTP response status codes indicate whether a specific HTTP request has been successfully completed.

```
print(resp)
print(resp.status_code)
```

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

- See some common codes:
  - 200: Success
  - 401: Unauthorized Error
  - 403: Forbidden
  - 404: Not Found

URL Params Values

- You may add parameter values to the HTTP request, e.g., page, date, language, type, sort…

```
requests.get(url, params={key: value})
```

- The parameter values must be in a dictionary.

It's a dictionary

```
url = 'https://ug.hkubs.hku.hk/course'
options = {
    'q' : 'iimt',
    'academic_year' : '2024-2025',
    'semester' : 'sem-1'
}
resp = requests.get(url, params = options)
print(resp.url)
```

https://ug.hkubs.hku.hk/course?q=iimt&academic_year=2024-2025&semester=sem-1

Other Optional Arguments

```
requests.get(url, params, args)
```

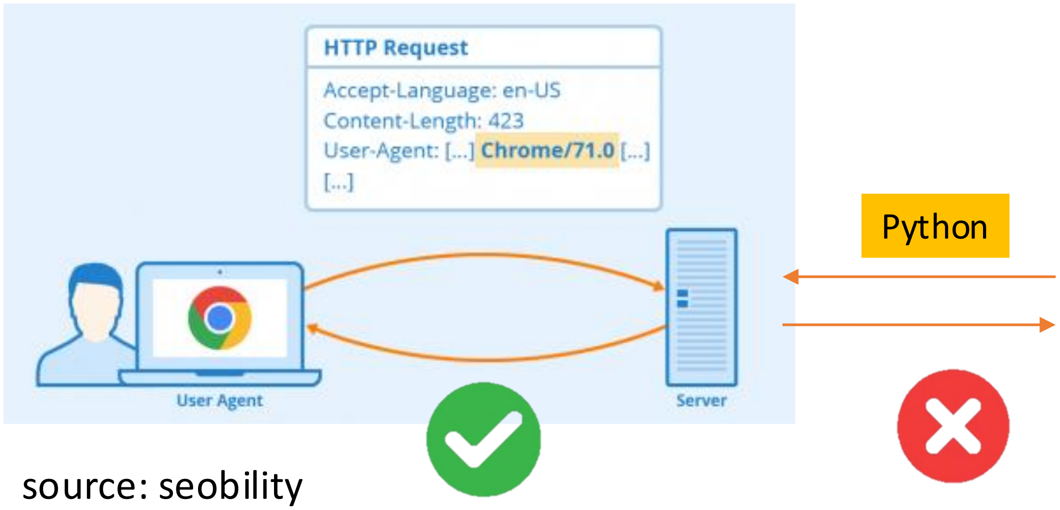
Argument	Description
allow_redirects	A Boolean to enable/disable redirection. Default True
auth	A tuple to enable a certain HTTP authentication. Default None
cert	A String or Tuple specifying a cert file or key. Default None
cookies	A dictionary of cookies to send to the specified url. Default None
headers	A dictionary of HTTP headers to send to the specified url. Default None
proxies	A dictionary of the protocol to the proxy url. Default None
stream	A Boolean indication if the response should be immediately downloaded (False) or streamed (True). Default False
timeout	A number, or a tuple, indicating how many seconds to wait for the client to make a connection and/or send a response. Default None which means the request will continue until the connection is closed
verify	A Boolean or a String indication to verify the servers TLS certificate or not. Default True

Bypass Anti-Spider

- Slow down scrawling, sleep for a few random seconds between requests.
- Change scrawling pattern.
- Change IPs.
- Use a user agent.
- Rotate user agent.
- Use APIs.
- …

User Agent

- A user agent is software that retrieves a web page from a server on the internet and displays it.
- A web browser is the most common user agent.
- When using Python as a user agent to make an HTTP request, the server is likely to deny your access.



```
url = 'http://www.amazon.com'
resp = requests.get(url)
```

source: seobility

User Agent

- Therefore, when using Python to make an HTTP request, we usually add headers to fake a web browser.

```
url = 'http://www.amazon.com'

headers = {
    'User-Agent' : 'Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/94.0.4606.71 Safari/537.36'
}
resp = requests.get(url, headers = headers)
```

It's a dictionary

- See a list of user agent:

  https://www.useragentstring.com/pages/Browserlist/

- Find one that works for you.

Retrieving the Page’s Source Code

Attribute	Description
text	Returns the content of the response, in Unicode (string)
content	Returns the content of the response, in bytes

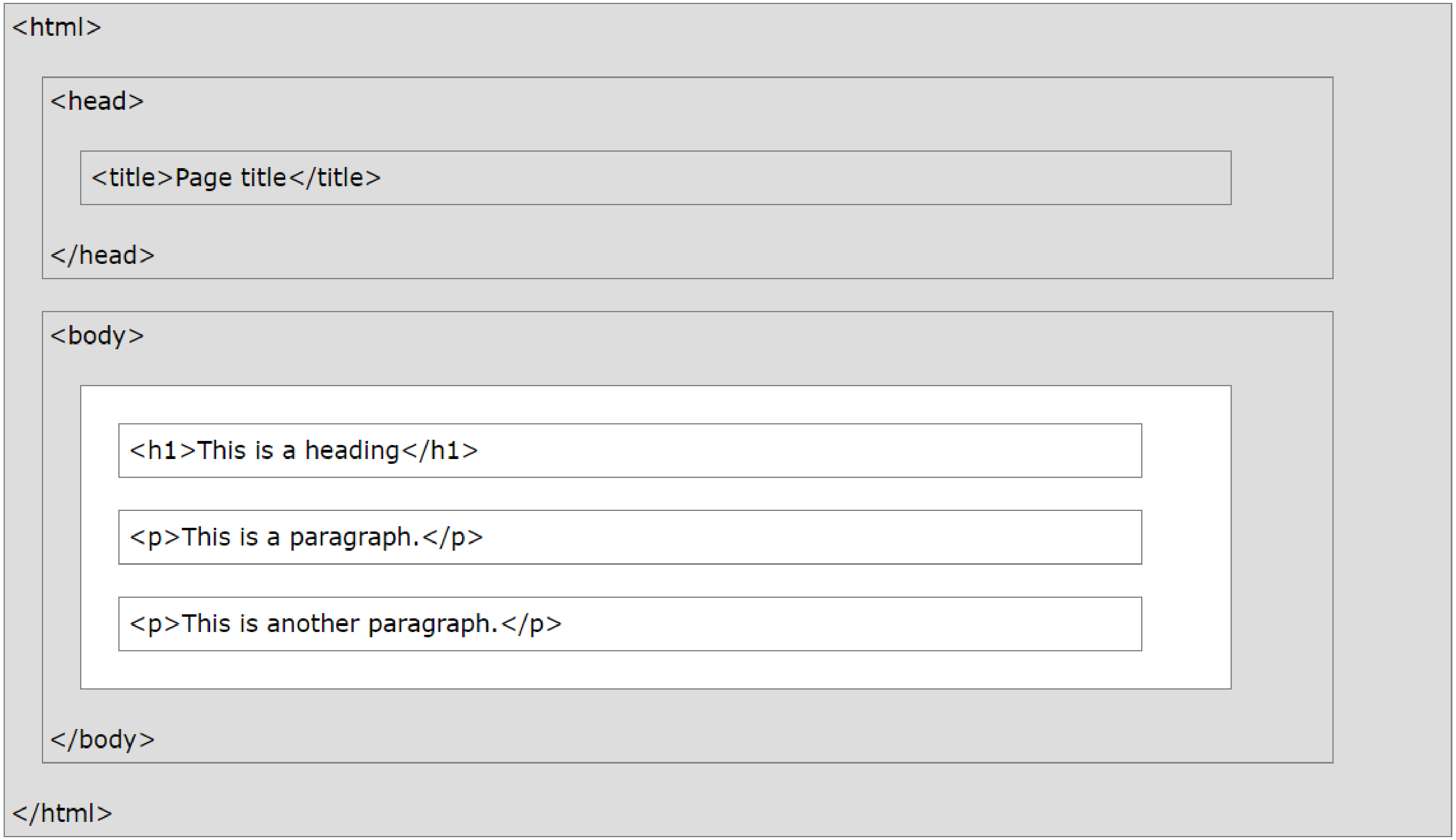
```
print(resp.text)
```

```
<!doctype html>
<html>
<head>
   <title>Example Domain</title>

   <meta charset="utf-8" />
   <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1" />
   <style type="text/css">
   body {
……
```

HTML Page

HTML Page Structure



```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
      <h1>This is a heading</h1>
      <p>This is a paragraph.</p>
      <p>This is another paragraph.</p>
  </body>
</html>
```

HTML Page Structure

- Web browsers use HTML (**H**yper**T**ext **M**arkup **L**anguage) to display webpages.
- Composed of elements. Elements are composed of a start tag and a closing tag.
- More details: https://www.w3schools.com/html/

```html
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Elements of Interest

- We are mostly NOT interested in <head>.
- The data we want to extract are generally found in elements under <body> such as:
  - A link
  - A table
  - An image
  - A list
  - A paragraph
  - A text
  - …

Links and Images

- Links are defined with the **<a>** tag with an attribute of href which is the URL.

```
<a href="https://www.w3schools.com/html/">
Visit our HTML tutorial</a>
```

Visit our HTML tutorial

- Images are defined with the **<img>** tag. There is no closing tag.
- The src attribute specifies the URL of the image:

```
<img src="url">
```

Table

- An HTML table is defined with the **<table>** tag.
- Each table row is defined with the **<tr>** tag.
- A table header is defined with the **<th>** tag. By default, table headings are bold and centered.
- A table data/cell is defined with the **<td>** tag.

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94
John	Doe	80

Unordered and Ordered Lists

- An unordered list starts with the **<ul>** tag. Each list item starts with the **<li>** tag.
- The list items will be marked with bullets by default:

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

•Coffee
•Tea
•Milk

- An ordered list has a type attribute in the **<ol>** tag.

```
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

1.Coffee
2.Tea
3.Milk

Type	Description
type="1"	by numbers (default)
type="A"	by uppercase letters
type="a"	by lowercase letters
type="I"	by uppercase roman numbers
type="i"	by lowercase roman numbers

Block and Inline Elements

- A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).
- The **<div>** element is a block-level element.

```
<div>Hello</div>
<div>World</div>
```

Hello
World

- An inline element does not start on a new line and only takes up as much width as necessary.
- The **<span>** element is an inline element.

```
<span>Hello</span>
<span>World</span>
```

Hello World

Attribute: Class

- The class attribute specifies one or more class names for an HTML element.
- The class name can be used by CSS and JavaScript to perform certain tasks for elements with the specified class name.
- Try here

```html
<h2 class="city">London</h2>
<p>London is the capital of
England.</p>

<h2 class="city">Paris</h2>
<p>Paris is the capital of
France.</p>

<h2 class="city">Tokyo</h2>
<p>Tokyo is the capital of
Japan.</p>
```

Attribute: ID

- The id attribute specifies a unique id for an HTML element (the value must be unique within the HTML document).
- *Note*: in reality, id may not be unique.
- [Try here]()

```
<!-- A unique element -->
<h1 id="myHeader">My Cities</h1>

<!-- Multiple similar elements -->
<h2 class="city">London</h2>
<p>London is the capital of
England.</p>

<h2 class="city">Paris</h2>
<p>Paris is the capital of
France.</p>

<h2 class="city">Tokyo</h2>
<p>Tokyo is the capital of
Japan.</p>
```

BeautifulSoup 4 (bs4)

A Sample Source Code of an HTML Page

```
html = '''
<html>
   <head><title>The King's story</title>
   </head>
   <body>
      <p class="title"><b>The King's story</b></p>
      <p class="story">Once upon a time there were five siblings; and their names were:
         <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
         <span>Meili</span>,
         <span class="brother">Eric</span>
         <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a>,
         <a class="sister" id="link3">Tillie</a>, and
         <a href="http://hku.hk/chao" class="brother" id="link4">Chao</a>,and they lived at
the bottom of a
         well.</p>
      <p class="story">...</p>
   </body>
</html>'''
```

Making the Soup

- BeautifulSoup supports the HTML parser included in Python's standard library, but it also supports a number of third-party Python parsers such as HTML5 and XML.
- Basic syntax

```
from bs4 import BeautifulSoup
BeautifulSoup(source_code, parser)
```

- The result is a BeautifulSoup object.

```
soup = BeautifulSoup(resp.text, 'html.parser')
```

Default value

```
type(soup)
```

bs4.BeautifulSoup

Finding Elements with Tag Names

- We can use a tag name to search in the tree. It returns a tag object.
- We can further call methods on the tag object.

`soup.title`	<title>The King’s story</title>
`soup.title.name`	'title'
`soup.title.string`	"The King’s story"
`soup.title.parent.name`	'head'
`soup.a`	<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>

Finding Elements with Soup Methods

Methods
find
findAll
find_all
findChild
findChildren
findNext
findNextSibling
fingParent
……

`find / find_all / findAll`

- `find` scans the entire document looking for an element. It returns the first element. If it can't find anything, returns `None`.

```
soup.find('title')
```

$\xleftrightarrow{\text{equivalent}}$

```
soup.head.title
```

- `find_all` / `findAll` return a list containing all the matched element(s). If they can't find anything, return an empty list.

```
soup.find_all('a')
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" id="link3">Tillie</a>,
 <a class="brother" href="http://hku.hk/chao" id="link4">Chao</a>]
```

find / find_all / findAll

- They also support other parameters:

```
find/find_all/findAll(name, attrs,
 recursive, string…)
```

  - attrs includes attributes like id, class, href, …
  - class is a reserved word in Python, use class_:

```
soup.find_all(id = 'link2')
```

```
soup.find_all('a', class_ = 'sister')
```

```
soup.find_all(href = re.compile("elsie"))
```

- Or, simply create a dictionary for attrs.

```
soup.find_all('a', attrs = {'class':'sister'})
```

Navigating the Tree With Methods

"next" tag(s)	"previous" tag(s)	child-parent
next	previous	childGenerator
nextGenerator	previousGenerator	children
nextSibling	previousSibling	parentGenerator
next_sibling	previous_sibling	parent
next_siblings	previous_siblings	parents
next_element	previous_element	
next_elements	previous_elements	

Extracting Attribute Values

```
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>
```

- We treat attributes like key-value pairs in a dictionary.

```
soup.a.attrs
```

```
{'href': 'http://example.com/elsie', 'class': ['sister'], 'id': 'link1'}
```

- Indexing the key (or using **`get`** method), obtain the values.

```
soup.a['href']
soup.a.get('href')
```

```
'http://example.com/elsie'
```

```
soup.a['id']
```

```
'link1'
```

Extracting Tag Content

Method	Description
text	Gets all strings within the tag block. Result is string.
get_text()	Gets all strings within the tag block. It allows for arguments. Result is string.
string	Gets all strings within the tag block. Result is NavigableString.

```
<p class="title"><b>The King's story</b></p>
```

tag_p

tag_b

```
tag_p.text
```

```
tag_b.text
```

"The King's story"

```
tag_p.get_text()
```

How to Scrape HTML pages

- Inspect the target (usually some text) in the page source*
- Understand the structure of the HTML page
- Break up your task into small pieces
- Print to see the tag structure of the small pieces
- Close in to your target element
- Extract and store the target text in a list or files

* For Safari users, turn on “developer mode” in setting.