MODULE 2.1

Web Automation

Program

- Opening pages in your web-browser
- Downloading web-pages with the request library
- Introduction to web-scraping
- Web-scraping with bs4
- Web automation with selenium

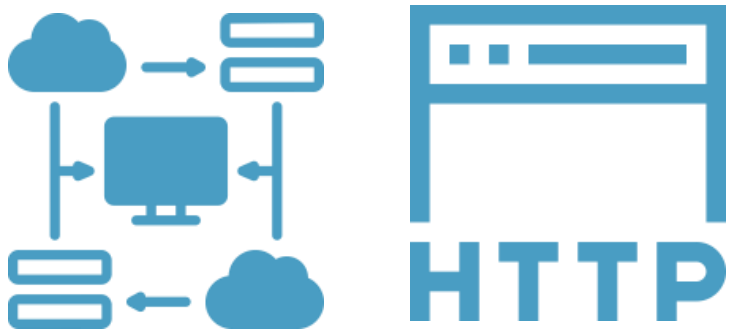
Learning Objectives

- You will be able **to (programmatically) open** specific webpages in your web browser
- You will be able **to (programmatically) send** HTTP requests with the requests library and to process the response
- You will be able **to parse** HTML pages with the beautiful soup (bs4) library
- You will be able **to (programmatically) control** your web browser - acting as
- human as possible - with the selenium library

Opening pages in your Web Browser

The HTTP Protocol

- The *HyperText Transport Protocol* is the set of rules designed to enable browsers to retrieve web documents from servers over the internet.
  - The dominant Application Layer Protocol on the Internet.
  - Invented for the Web - to retrieve HTML, images, documents, etc.
  - Extended to be data in addition to documents - RSS, Web Services, etc..
  - Basic Concept: Make a Connection - Request a document - Retrieve the Document - Close the Connection.



The HTTP Protocol

Characteristic	Description
Protocol type	Application layer protocol
Purpose	Used for transferring hypertext (text, images, videos, etc.)
Request/Response	Client sends a request to a server, and server responds with a response
Methods	GET, POST, PUT, DELETE, HEAD, OPTIONS, CONNECT, TRACE, PATCH
URI (Uniform Resource Identifier)	Identifies the resource (e.g., URL for web addresses)
Headers	Used to convey additional information in the request or response
Status codes	Three-digit codes in response indicating the outcome of the request (e.g., 200 OK, 404 Not Found)

The webbrowser library

- The webbrowser library in Python is a very basic built-in module that provides a high-level interface for working with web browsers.
  - It allows you to open URLs in web browsers, control the behavior of web browsers, and perform basic web-related tasks programmatically from within a Python script.
  - Provides functions to open URLs in the default web browser, a specific web browser, or a new web browser window/tab. It can also be used to display HTML content, search for a query in a web browser, and retrieve the current URL from a web browser.

Example...

```
>>> import webbrowser
>>> websites.ls
{'lms': 'https://lms-iai.cyberpro-israel.org/login/index.php', 'cywaria':
'https://azcybersecuritycenter.cywaria.net/#/login', 'mail':
'https://srv.cert.az:2096/'}
>>> for i in websites.ls.values():
...     webbrowser.open(i)
```

Can you think of other use cases for this very basic `webbrowser` library?

Discussion

Downloading web-pages with the request library

The requests library

- The requests library is a popular Python library for making HTTP requests and handling responses.
- It provides a simple and convenient way to interact with web APIs, send HTTP requests such as GET, POST, PUT, DELETE, and more, and handle the responses in a flexible and efficient manner.
- The library may need to be installed with pip first
- The requests module was written because Python's urllib2 module is considered too complicated to use.

Example...

```
>>> import requests
>>> url = 'https://en.wikipedia.org/wiki/%22Hello,_World!%22_program'
>>> result = requests.get(url)
>>> type(result)
<class 'requests.models.Response'>
>>> dir(result)
['__attrs__', '__bool__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__enter__', '__eq__', '__exit__', '__format__', '__ge__', '__getattribute__',
'__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__nonzero__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', '_content', '_content_consumed', '_next', 'apparent_encoding', 'close',
'connection', 'content', 'cookies', 'elapsed', 'encoding', 'headers', 'history',
'is_permanent_redirect', 'is_redirect', 'iter_content', 'iter_lines', 'json', 'links',
'next', 'ok', 'raise_for_status', 'raw', 'reason', 'request', 'status_code', 'text', 'url']
```

Introduction to 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 scrape web pages - we call this “spidering the web” or “web crawling”.

Use Cases of Scraping

- Pull intelligence data - particularly social data - who links to who?
- Get your own data back out of some system that has no “export capability”
- Monitor sites for new information
- Spider the web to make a database for a search engine
- And many more…

What can be a point of concern about scraping the web?

Discussion

Legal Considerations

- It's important to note that web scraping for cybersecurity purposes should always be conducted in *compliance* with applicable laws and regulations, and with proper authorization from website owners or data owners.
- Ethical considerations, data privacy, and legal requirements should be carefully considered and followed when conducting web scraping activities for cybersecurity purposes.
- Republishing copyrighted information is not allowed.
- Violating terms of service is not allowed.

User Conduct

You understand that except for advertising programs offered by us on the Site (e.g., Facebook Flyers, Facebook Marketplace), the Service and the Site are available for your personal, non-commercial use only. You represent, warrant and agree that no materials of any kind submitted through your account or otherwise posted, transmitted, or shared by you on or through the Service will violate or infringe upon the rights of any third party, including copyright, trademark, privacy, publicity or other personal or proprietary rights; or contain libelous, defamatory or otherwise unlawful material.

In addition, you agree not to use the Service or the Site to:

- harvest or collect email addresses or other contact information of other users from the Service or the Site by electronic or other means for the purposes of sending unsolicited emails or other unsolicited communications;
- use the Service or the Site in any unlawful manner or in any other manner that could damage, disable, overburden or impair the Site;
- use automated scripts to collect information from or otherwise interact with the Service or the Site;

http://www.facebook.com/terms.php

Web scraping with bs4

The HTML Format

- *Hypertext Markup Language (HTML)* is the format that web pages are written in.
- Learning resources:
  - https://developer.mozilla.org/en-US/learn/html/
  - https://htmldog.com/guides/html/beginner/
  - https://www.codecademy.com/learn/learn-html

Why is it not a good idea to parse html with the `re` library?

Discussion

The Beautiful Soup Library

- Beautiful Soup (**bs4**) is a popular Python library used for web scraping, data extraction, and HTML parsing tasks. It provides a convenient way to navigate and search through the HTML or XML structure of a web page, and extract the data you need for further processing or analysis.
- It is a much better option than parsing html pages with regular expressions
- The library may need to be installed with `pip` first

Example…

```
>>> import requests, bs4
>>> url='https://en.wikipedia.org/wiki/%22Hello,_World!%22_program'
>>> result=requests.get(url)
>>> soup=bs4.BeautifulSoup(result.text, 'html.parser')
>>> type(soup)
<class 'bs4.BeautifulSoup’>
>>> dir(soup)
… snip …
>>> h2_list = soup.select(‘h2’)
>>> str(h2_list[0])
'<h2 class="vector-pinnable-header-label">Contents</h2>’
>>> h2_list[1].getText()
'History[edit]'
```

Selector Statement In select()

Selector statement	Description
tagname	Selects all occurrences of the specified HTML tag, e.g., select('p') selects all <p> tags.
.class	Selects all occurrences of the specified CSS class, e.g., select('.container') selects all elements with class="container".
#id	Selects the element with the specified HTML id attribute, e.g., select('#header') selects the element with id="header".
tagname.class	Selects all occurrences of the specified HTML tag with the specified CSS class, e.g., select('div.container') selects all tags with class="container".
tagname#id	Selects the element with the specified HTML tag and id attribute, e.g., select('input#username') selects the <input> tag with id="username".
tagname[attr=value]	Selects all occurrences of the specified HTML tag with the specified attribute and value, e.g., select('a[href="https://example.com"]') selects all <a> tags with href="https://example.com"

Web automation with `selenium`

Web Automation With selenium

- The `selenium` module lets Python directly control the browser by programmatically clicking links and filling in login information, almost as though there were a human user interacting with the page.
- You can interact with web pages in a much more advanced way than with requests and bs4; but because it launches a real web browser,
  - It is a bit slower and hard to run in the background if, say, you just need to download some files from the web.
  - It can circumvent preventive measures from websites
  - It can pass for “human” more easily
  - user-agent

Controlling The Browser

```
kali@kali:~$ pip install –user selenium
kali@kali:~$ python3

>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> type(browser)
<class ‘selenium.webdriver.firefox.webdriver.WebDriver’>
>>> url=‘https://en.wikipedia.org/wiki/%22Hello,_World!%22_program
>>> browser.get(url)
```

Finding Elements

Method Name	Description
find_element_by_id(id)	Finds and returns the first element with the specified id attribute.
find_element_by_name(name)	Finds and returns the first element with the specified name attribute.
find_element_by_class_name(class_name)	Finds and returns the first element with the specified class attribute.
find_element_by_tag_name(tag_name)	Finds and returns the first element with the specified HTML tag name.
find_element_by_css_selector(css_selector)	Finds and returns the first element that matches the specified CSS selector.
find_element_by_xpath(xpath)	Finds and returns the first element that matches the specified XPath expression.
find_element_by_link_text(link_text)	Finds and returns the first anchor (<a>) element whose visible text matches the specified link_text.
find_element_by_partial_link_text(partial_link_text)	Finds and returns the first anchor (<a>) element whose visible text partially matches the specified partial_link_text.
find_elements_by_*()	Similar to the above methods, but returns a list of all matching elements instead of just the first one.

The WebElement Object

- The finding methods return a WebElement Object with the following properties and methods

Attribute or method	Description
tag_name	The tag name, such as 'a' for an <a> element
get_attribute(*name*)	The value for the element's name attribute
text	The text within the element, such as 'hello' in hello
clear()	For text field or text area elements, clears the text typed into it
is_displayed()	Returns True if the element is visible; otherwise returns False
is_enabled()	For input elements, returns True if the element is enabled; otherwise returns False
is_selected()	For checkbox or radio button elements, returns True if the element is selected; otherwise returns False
location	A dictionary with keys 'x' and 'y' for the position of the element in the page
click()	Clicks on the element.
send_keys()	Sends keystrokes to the element.

30 minutes exercise…

- Create a script that takes in your username for the LMS from the commandline (use `sys.arv`)
- Then it opens up the LMS in the selenium browser and autofill your username
- Then it prompt for your password and autofill your password
- Then it ‘clicks’ the login button to login

- **Note**: never hardcode your password in scripts!

Other Useful Options

- The `selenium` module can simulate clicks on various browser buttons as well through the following methods:
  - `browser.back()`- Clicks the Back button.
  - `browser.forward()` - Clicks the Forward button.
  - `browser.refresh()` - Clicks the Refresh/Reload button.
  - `browser.quit()` - Clicks the Close Window button.
- With the `selenium.webdriver.common.keys` you can also send key presses to the browser

Learning Objectives

- You will be able **to (programmatically) open** specific webpages in your web browser
- You will be able **to (programmatically) send** HTTP requests with the requests library and to process the response
- You will be able **to parse** HTML pages with the beautiful soup (bs4) library
- You will be able **to (programmatically) control** your web browser - acting as
- human as possible - with the selenium library