Part 1 Web Scraping

Web Scraping is an art where one has to study the website and work according to the dynamics of that particular website.

Most common tools used for web scraping in python are demonstrated below.

1. requests https://requests.readthedocs.io/en/latest/
2. beautiful soup https://beautiful-soup-4.readthedocs.io/en/latest/
3. Selenium https://selenium-python.readthedocs.io/
4. Scrapy https://docs.scrapy.org/en/latest/

We will be working on the first three and the fourth one can be explored in the homeworks.

We will be scraping 4 websites today:

1. ScrapeThisSite
2. GeeksforGeeks
3. CNBC
4. Hoopshype

There are different techniques to be used when scraping a dynamic website vs a static website which will be discussed in the coming sections

Some websites have their APIs open and those can be used to directly fetch the data without the need of scraping the HTML or XML pages.

```
# installing the libraries
!pip install requests
!pip install bs4
!pip install selenium
!apt-get update
!apt install chromium-chromedriver
!cp /usr/lib/chromium-browser/chromedriver /usr/bin
```

Show hidden output

```
# importing the libraries
import requests
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import pandas as pd
import json
from google.colab import drive
import sys
```

```
drive.mount("/content/drive/")
```

Show hidden output

```
url = 'https://www.scrapethissite.com/pages/forms/'
```

```
page = requests.get(url)
```

```
soup = BeautifulSoup(page.text, 'html')
```

```
print(soup)
```

Show hidden output

```
soup.find('div')
```

Show hidden output

```
soup.find_all('p')
```

Show hidden output

```
soup.find('p', class_ = 'lead').text
```

Show hidden output

```
soup.find('p', class_ = 'lead').text.strip()
```

Show hidden output

```
soup.find_all('th')
```

Show hidden output

```
soup.find('th').text.strip()
```

Show hidden output

My work for Task 1 and Task 5:

I created a pandas dataframe with the table available on the website. This is for multiple tasks i.e For Task 1 and Task 5. I have discussed this with the professor and she said that is fine.

```
import pandas as pd

rows = []
for i in range(1,7):
  url = 'https://www.scrapethissite.com/pages/forms/?page_num={}&per_page=100'.format(i)
  print(url)
  page = requests.get(url)
  soup = BeautifulSoup(page.text, 'html')
  table = soup.find('table')
  columns = [th.text.replace("\n","").strip() for th in table.find_all('th')]
  for tr in table.find_all('tr')[1:]:
    temp = []
    for td in tr.find_all('td'):
      x = td.text.replace("\n","").strip()
      temp.append(x)
    rows = rows + [temp]
  # print(len(rows))
  # print(len(set(tuple(row) for row in rows)))

df_from_html = pd.DataFrame(rows, columns=columns)
df_from_html
```

https://www.scrapethissite.com/pages/forms/?page_num=1&per_page=100
https://www.scrapethissite.com/pages/forms/?page_num=2&per_page=100
https://www.scrapethissite.com/pages/forms/?page_num=3&per_page=100
https://www.scrapethissite.com/pages/forms/?page_num=4&per_page=100
https://www.scrapethissite.com/pages/forms/?page_num=5&per_page=100
https://www.scrapethissite.com/pages/forms/?page_num=6&per_page=100

	Team Name	Year	Wins	Losses	OT Losses	Win %	Goals For (GF)	Goals Against (GA)	+ / -
0	Boston Bruins	1990	44	24		0.55	299	264	35
1	Buffalo Sabres	1990	31	30		0.388	292	278	14
2	Calgary Flames	1990	46	26		0.575	344	263	81
3	Chicago Blackhawks	1990	49	23		0.613	284	211	73
4	Detroit Red Wings	1990	34	38		0.425	273	298	-25
...	...	...	...	...	...	...	...	...	...
577	Tampa Bay Lightning	2011	38	36	8	0.463	235	281	-46
578	Toronto Maple Leafs	2011	35	37	10	0.427	231	264	-33
579	Vancouver Canucks	2011	51	22	9	0.622	249	198	51
580	Washington Capitals	2011	42	32	8	0.512	222	230	-8
581	Winnipeg Jets	2011	37	35	10	0.451	225	246	-21

582 rows × 9 columns

```
# getting the first URL
# open the URL in parallel in other tab to check the information we are extracting
url = "https://www.geeksforgeeks.org/python-programming-language/"
```

```
  # hitting the url and getting the response
  res = requests.get(url)
  print(res.status_code)
```

Show hidden output

```
# creating a soup object from the returned html page
sp = BeautifulSoup(res.text, "lxml")
sp
```

Show hidden output

```
# printing it in readable format
print(sp.prettify())
```

Show hidden output

```
# parsing title element of the page
print(sp.title)
print(sp.title.name)
print(sp.title.string)
print(sp.title.parent.name)
```

Show hidden output

```
# extracting the title of the article
print(sp.find("div", {"class" : "article-title"}).text)
```

Show hidden output

```
print(sp.find("div"))
```

Show hidden output

```
  # extracting the date of the article
  date_label = sp.find('span', class_='strong', text='Last Updated : ')
  date_span = date_label.find_next('span')
  last_updated_date = date_span.text
  print(f"Last Updated Date: {last_updated_date}")

```

Show hidden output

```
# extracting the content of the article
# it extracts everyhting together, in the next sections we can see how to iteratively extract information paragraph by paragraph
print(sp.find("div", {"class" : "text"}).text.strip().replace("\n", " "))
```

Show hidden output

```
# extracting the links found in the bottom of the article for further reading
for tag in sp.find("div", {"class":"gfg-similar-reads-list"}).findAll("a", href = True): print(tag.text, "\n", tag["href"], "\n")
```

Show hidden output

```
# extracting the links found in the bottom of the article for further reading
for link in sp.find('div', class_='gfg-similar-reads-list').find_all('a'):
    print(link.get('href'))
```

Show hidden output

```
# extracting information paragraph by paragraph
for tag in sp.find("div", {"class":"text"}).findAll("p"): print(tag.text)
```

Show hidden output

Almost all the geeksforgeeks articles have the same format and hence all of them can be scraped using the same code, this repeatability is useful while doing web scraping as a block of code can help one get a lot of information in a structured way

Other information can be extracted using similar methods as used in geeksforgeeks, this can be done in homework

Again same as geeksforgeeks, all articles of scrapethissite are similar in structure and hence the same code can be used to scrap through all the articles of this website

Now working with a dynamic website that has its API open.

Open the CNBC website and search for any topic. If we search for SPORTS the URL looks like this: https://www.cnbc.com/search/?query=SPORTS&qsearchterm=SPORTS and if we search for POLITICS the URL looks like this: https://www.cnbc.com/search/?query=POLITICS&qsearchterm=POLITICS

Here we can observe a pattern and we can predict what the url would look like if we search something else, this information can be used for reusability and repseatability of code.

Now we can see that there are more than 50,000 results for the topic politics but only 10 are loaded in the beginning. Once we scroll down, next 10 are loaded and so on. To load the next results when the user scrolls down, an API is hit and link to that API is found from the network section of the inspect element: https://api.queryly.com/cnbc/json.aspx?queryly_key=31a35d40a9a64ab3&query=POLITICS&endindex=10&batchsize=10&callback=&showfaceted=false&timezoneoffset=420&facetedfields=formats&facetedkey=formats%7C&facetedvalue=!Press%20Release%7C&additionalindexes=4cd6f71fbf22424d,937d600b0d0d4e23,3bfbe40caee7443e,626fdfcd96444f28

Here playing around with the endindex and batchsize parameter we can get various results.

Now the response of this API call would be a JSON and hence the need for parsing a web page is gone when there is an open API

```
# getting the third url
url = "https://api.queryly.com/cnbc/json.aspx?queryly_key=31a35d40a9a64ab3&query=POLITICS&endindex=10&batchsize=10&callback=&showfaceted=
```

```
# hitting the url and getting the response
res = requests.get(url)
print(res.status_code)
```

Show hidden output

```
res.text
```

Show hidden output

```
# getting the description of each news article
for description in json.loads(res.text)["results"]: print(description["description"], "\n")
```

Show hidden output

Finally working with Selenium

```
!echo | sudo add-apt-repository ppa:saiarcot895/chromium-beta
!sudo apt remove chromium-browser
!sudo snap remove chromium
!sudo apt install chromium-browser -qq
# Chromium (an open-source version of Chrome) and Chromium WebDriver (which allows Selenium to control Chromium).

```

Show hidden output

```
!pip3 install selenium --quiet
!apt-get update
!apt install chromium-chromedriver -qq
!cp /usr/lib/chromium-browser/chromedriver /usr/bin
#Selenium requires a browser driver (in this case, chromedriver) to communicate with the browser. You're installing it using the chromium
```

Show hidden output

```
!pip install --upgrade selenium
```

Show hidden output

```
!pip install selenium
!apt-get update
!apt-get install -y chromium-chromedriver
```

Show hidden output

Task 2:

Salaries of players

```
import sys
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.support.ui import WebDriverWait

sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')
# download the selenium chromedriver executable file and paste the link in the following code
# this code should open a new chrome window in your machine
sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')
chrome_service = ChromeService(
    executable_path='/usr/lib/chromium-browser/chromedriver',
    log_path='/dev/null'  # You can change the log path as needed
)
driver = webdriver.Chrome(service=chrome_service,options=chrome_options)
#The ChromeService class sets up the path to the chromedriver
```

```
# this code should open hoopshype website in your newly opened chrome window
driver.get('https://hoopshype.com/salaries/players/')
# After setting up the Selenium WebDriver, the browser (in headless mode) navigates to HoopsHype Salaries page
```

```
# getting players name list
# wait = WebDriverWait(driver, 10)
# element = wait.until(driver.find_elements("xpath", '//td[@class="name"]'))
players = driver.find_elements("xpath", '//td[@class="name"]')
#uses an XPath selector to find all the HTML elements on the page that have a class attribute "name", which corresponds to the player nam
```

```
players_list = []
try:
  for p in range(len(players)):
    players_list.append(players[p].text)
except:
  print("Stale Element Error")
print(len(set(players_list)))
players_list = set(players_list)
```

```
600
```

```
  salaries = []
  for i in range(len(players_list)-2):
    salary = driver.find_elements("xpath", '//*[@id="content-container"]/div/div[3]/div[2]/div[2]/div[1]/table/tbody/tr[{}]/td[3]'.format(
    salaries.append(salary[0].text)
  salaries
```

```
['$55,761,217',
 '$51,415,938',
 '$51,415,938',
 '$51,179,020',
 '$50,203,930',
 '$49,205,800',
 '$49,205,800',
 '$49,205,800',
 '$49,205,800',
 '$49,205,800',
 '$48,798,677',
 '$48,787,676',
```

```
    '$48,787,676',
    '$48,728,845',
    '$43,827,586',
    '$43,219,440',
    '$43,031,940',
    '$43,031,940',
    '$43,031,940',
    '$42,846,615',
    '$42,176,400',
    '$42,176,400',
    '$42,176,400',
    '$42,176,400',
    '$41,000,000',
    '$40,500,000',
    '$40,338,144',
    '$36,725,670',
    '$36,725,670',
    '$36,725,670',
    '$36,637,932',
    '$36,016,200',
    '$36,016,200',
    '$35,859,950',
    '$35,859,950',
    '$35,410,310',
    '$35,147,000',
    '$35,147,000',
    '$34,848,340',
    '$34,848,340',
    '$34,848,340',
    '$34,005,250',
    '$34,005,126',
    '$33,653,846',
    '$33,333,333',
    '$32,500,000',
    '$31,666,667',
    '$30,000,000',
    '$30,000,000',
    '$29,793,104',
    '$29,651,786',
    '$29,347,826',
    '$29,283,801',
    '$29,268,293',
    '$29,000,000',
    '$28,939,680',
    '$27,556,817',
    '$27,173,913'
```

Task 3

Website 1

```
# Apart from that choose any 2 websites of your choice and extract meaningful and structured information from there.
import sys
from selenium.webdriver.chrome.service import Service as ChromeService
sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')
# download the selenium chromedriver executable file and paste the link in the following code
# this code should open a new chrome window in your machine
sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')
chrome_service = ChromeService(
    executable_path='/usr/lib/chromium-browser/chromedriver',
    log_path='/dev/null'  # You can change the log path as needed
)
driver = webdriver.Chrome(service=chrome_service,options=chrome_options)
#The ChromeService class sets up the path to the chromedriver
```

```
# this code should open hoopshype website in your newly opened chrome window
driver.get('https://www.learnpytorch.io/')
# After setting up the Selenium WebDriver, the browser (in headless mode) navigates to HoopsHype Salaries page
```

```
# getting players name list
main_content = driver.find_elements("xpath", '/html/body/div[3]/main/div/div[3]')
#uses an XPath selector to find all the HTML elements on the page that have a class attribute "name", which corresponds to the player nam
```

```
main_content[0].text
```

```
'Learn PyTorch for Deep Learning: Zero to Mastery book\nWelcome to the second best place on the internet to learn PyTorch (the first be
ing the PyTorch documentation).\nThis is the online book version of the Learn PyTorch for Deep Learning: Zero to Mastery course.\nThis
course will teach you the foundations of machine learning and deep learning with PyTorch (a machine learning framework written in Pytho
n).\nThe course is video based. However, the videos are based on the contents of this online book.\nFor full code and resources see the
course GitHub.\nOtherwise, you can find more about the course below.\nDoes this course cover PyTorch 2.0?\nYes. PyTorch 2.0 is an addit
ive release to previous versions of PyTorch.\nThis means it adds new features on top of the existing baseline features of PyTorch.\nThi
```

```
# getting all list items from the unordered list
list_of_topics = driver.find_elements("xpath", '/html/body/div[3]/main/div/div[1]/div/div/nav/ul/li/a/span')

# # Fetching text using 'innerText'
for topic in list_of_topics:
    text = topic.get_attribute('innerText').strip()
    if text:
        print(text)

```

```
Home
00. PyTorch Fundamentals
01. PyTorch Workflow Fundamentals
02. PyTorch Neural Network Classification
03. PyTorch Computer Vision
04. PyTorch Custom Datasets
05. PyTorch Going Modular
06. PyTorch Transfer Learning
07. PyTorch Experiment Tracking
08. PyTorch Paper Replicating
09. PyTorch Model Deployment
A Quick PyTorch 2.0 Tutorial
PyTorch Extra Resources
PyTorch Cheatsheet
The Three Most Common Errors in PyTorch
```

Similarly other information such as players' salaries can also be easily extracted and can be done as homework

Website 2

```
# Apart from that choose any 2 websites of your choice and extract meaningful and structured information from there.
import sys
from selenium.webdriver.chrome.service import Service as ChromeService
sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')
# download the selenium chromedriver executable file and paste the link in the following code
# this code should open a new chrome window in your machine
sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')
chrome_service = ChromeService(
    executable_path='/usr/lib/chromium-browser/chromedriver',
    log_path='/dev/null'  # You can change the log path as needed
)
driver = webdriver.Chrome(service=chrome_service,options=chrome_options)
#The ChromeService class sets up the path to the chromedriver
```

```
# this code should open hoopshype website in your newly opened chrome window
driver.get('https://en.wikipedia.org/wiki/Main_Page')
# After setting up the Selenium WebDriver, the browser (in headless mode) navigates to HoopsHype Salaries page
```

```
# getting players name list
main_content = driver.find_elements("xpath", '//*[@id="mp-tfa"]/p')
#uses an XPath selector to find all the HTML elements on the page that have a class attribute "name", which corresponds to the player nam
main_content[0].text
```

```
'An attempted coup took place on September 13, 1964, in South Vietnam against the ruling military junta, led by Nguyễn Khánh (picture
d). In the preceding month, Khánh had tried to improve his leadership by declaring a state of emergency, provoking protests and riots.
He made concessions to the protesters and removed military officials linked to former President Ngo Dinh Diem, including Lâm Văn Phát a
nd Dương Văn Đức. They responded with a coup, broadcasting their promise to revive Diem's policies. Khánh evaded capture and rallied al
lies while the U.S. continued their support for his rule. Khánh forced Phát and Đức to capitulate the next morning and various coup lea
ders appeared at a media conference where they denied that a coup had taken place. To maintain power, Khánh tried to court support from
```

Task 4

```
!pip install scrapy
!pip install pyppeteer
!pip install scrapy asyncio
```

Show hidden output

```
import scrapy
from scrapy.crawler import CrawlerProcess

class BasicSpider(scrapy.Spider):
    name = "basic"
    start_urls = ['https://www.learnpytorch.io/']

    def parse(self, response):
        for title in response.css('h1::text'):
            data = {'title': title.get()}
            print(data)
            yield data

        next_page = response.css('a.next::attr(href)').get()
        if next_page is not None:
            yield response.follow(next_page, self.parse)

process = CrawlerProcess(settings={
    "FEEDS": {
        "output.json": {"format": "json"},  # Save data to a JSON file
    },
})

process.crawl(BasicSpider)
process.start()
```

```
 'log_count/DEBUG': 3,
 'log_count/INFO': 11,
 'memusage/max': 211329024,
 'memusage/startup': 211329024,
 'response_received_count': 1,
 'scheduler/dequeued': 1,
 'scheduler/dequeued/memory': 1,
 'scheduler/enqueued': 1,
 'scheduler/enqueued/memory': 1,
 'start_time': datetime.datetime(2024, 9, 13, 1, 29, 35, 765763, tzinfo=datetime.timezone.utc)}
INFO:scrapy.core.engine:Spider closed (finished)
2024-09-13 01:29:36 [scrapy.core.engine] INFO: Spider closed (finished)
{'title': 'Learn PyTorch for Deep Learning: Zero to Mastery book'}
```

Part 2: PPT Scraping

```
from google.colab import drive
drive.mount('/content/drive')
```

Show hidden output

```
!pip install python-pptx
```

Show hidden output

Extracting text from PPT using the python-pptx library, it is typically used for generating ppts from databases but we can exploit some of its features here to extract text from ppts, this is a very basic example and it can be explored further as per the need. documentation to the libary: https://python-pptx.readthedocs.io/en/latest/

```
# importing library
from pptx import Presentation
```

```
# extracting texts slide wise and section wise
# open the PPT in parallel to check the outcome
prs = Presentation("/content/drive/MyDrive/Natural Language Processing/HW2/Your big idea.pptx")
counter_slide = 1
for slide in prs.slides:
    print("slide:", counter_slide, "\n")
    counter_content = 1
    for shape in slide.shapes:
        try:
            print("content:", counter_content, shape.text, "\n")
            counter_content += 1
        except: continue
    print("\n\n")
    counter_slide += 1
```

Show hidden output

Similarly other components of the PPT can be extracted after following the documentation as per need

Task 6

Extracting table from PPT using pptx

```
# extracting texts slide wise and section wise
# open the PPT in parallel to check the outcome
prs = Presentation("/content/drive/MyDrive/Natural Language Processing/HW2/Scrape This.pptx"
counter_slide = 1
for slide in prs.slides:
    print("slide:", counter_slide, "\n")
    counter_content = 1
    for shape in slide.shapes:
      if(shape.has_text_frame):
        print("content:", counter_content, shape.text, "\n")
        counter_content += 1

      if shape.has_table:
```

```
        table = shape.table
        for row_number, row in enumerate(table.rows, start=1):
            row_data = []
            for cell in row.cells:
                row_data.append(cell.text)
            print(f"Row {row_number}: {row_data}")

    print()
    counter slide += 1
```

```
slide: 1

content: 1 Scrape This

content: 2


slide: 2

Row 1: ['Team Name', 'Year', 'Wins', 'Losses', 'OT Losses', 'Win %', 'Goals For (GF)', 'Goals Against (GA)', '+ / -']
Row 2: ['Boston Bruins', '1990', '44', '24', '', '0.55', '299', '264', '35']
Row 3: ['Buffalo Sabres', '1990', '31', '30', '', '0.388', '292', '278', '14']
Row 4: ['Calgary Flames', '1990', '46', '26', '', '0.575', '344', '263', '81']
Row 5: ['Chicago Blackhawks', '1990', '49', '23', '', '0.613', '284', '211', '73']
Row 6: ['Detroit Red Wings', '1990', '34', '38', '', '0.425', '273', '298', '-25']
Row 7: ['Edmonton Oilers', '1990', '37', '37', '', '0.463', '272', '272', '0']
```

Part 3: PDF Scraping

[ ] ↳ *8 cells hidden*

Homework

1. As discussed in the demo above, using the example of geeksforgeeks extract the information from ScrapethisSite articles apart from those that is already demonstrated.
2. As discussed in the demo above, extract the salaries of each of the players from the hoopshype website using the example of how to extract the names.
3. Apart from that choose any 2 websites of your choice and extract meaningful and structured information from there.
4. Also explore the scrapy library to perform webscraping apart from the three discussed above in the demo
5. Pick a website that has tabular data and try to scrap it using the tools studied during the demo.

(The datasets you will be collecting for the projects would be by text extraction so make sure to extract usable structured information)

6. Extract table from a PPT using the same library.