1. Explain various steps involved in Webscraping framework and illustrate it using Beautiful Soup. (URl: www.quotestoscrape.com) (TLO: 1.2) (6M)

Steps involved in web scraping:

Step 1: Installing the required third-party libraries (1M)

- **pip** is a package management system used to install and manage software packages written in Python.:

```
pip install requests

pip install html5lib

pip install bs4
```

Step 2: Accessing the HTML content from webpage (1M)

filter_none

brightness_5

```
import requests

URL = "https://www.geeksforgeeks.org/data-structures/"

r = requests.get(URL)

print(r.content)
```

Let us try to understand this piece of code.

- First of all import the requests library.
- Then, specify the URL of the webpage you want to scrape.
- Send a HTTP request to the specified URL and save the response from server in a response object called r.
- Now, as print r.content to get the **raw HTML content** of the webpage. It is of 'string' type.

Step 3: Parsing the HTML content (2M)

filter_none

brightness_5

```
#This will not run on online IDE

import requests

from bs4 import BeautifulSoup


URL = "http://www.values.com/inspirational-quotes"

r = requests.get(URL)


soup = BeautifulSoup(r.content, 'html5lib') # If this line causes an error, run 'pip ins

print(soup.prettify())
```

the BeautifulSoup library is built on the top of the HTML parsing libraries like html5lib, lxml, html.parser, etc. So BeautifulSoup object and specify the parser library can be created at the same time.

In the example above,

```
soup = BeautifulSoup(r.content, 'html5lib')
```

We create a BeautifulSoup object by passing two arguments:

- **r.content** : It is the raw HTML content.
- **html5lib** : Specifying the HTML parser we want to use.

Now **soup.prettify()** is printed, it gives the visual representation of the parse tree created from the raw HTML content.

Step 4: Searching and navigating through the parse tree (2 M)

The soup object contains all the data in the nested structure which could be programmatically extracted.

In our example, we are scraping a webpage consisting of some quotes.

So, we would like to create a program to save those quotes (and all relevant information about them).

filter_none

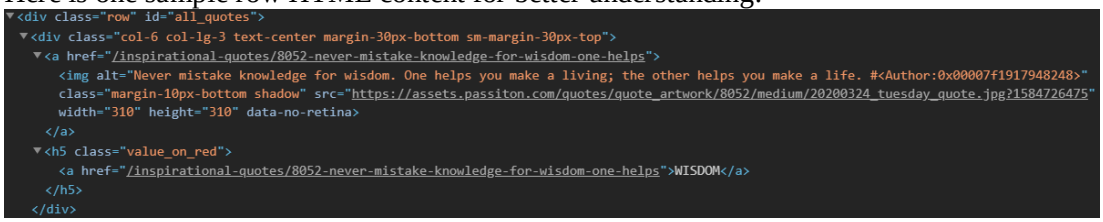
brightness_5

```
#Python program to scrape website

#and save quotes from website

import requests

from bs4 import BeautifulSoup

import csv


URL = "http://www.values.com/inspirational-quotes"

r = requests.get(URL)


soup = BeautifulSoup(r.content, 'html5lib')


quotes=[]  # a list to store quotes


table = soup.find('div', attrs = {'id':'all_quotes'})


for row in table.findAll('div',

                         attrs = {'class':'col-6 col-lg-3 text-center margin-30px-bottom
```

```
    quote = {}
    quote['theme'] = row.h5.text
    quote['url'] = row.a['href']
    quote['img'] = row.img['src']
    quote['lines'] = row.img['alt'].split(" #")[0]
    quote['author'] = row.img['alt'].split(" #")[1]
    quotes.append(quote)


filename = 'inspirational_quotes.csv'
with open(filename, 'w', newline='') as f:
    w = csv.DictWriter(f,['theme','url','img','lines','author'])
    w.writeheader()
    for quote in quotes:
        w.writerow(quote)
```

Before moving on, we recommend you to go through the HTML content of the webpage which we printed using soup.prettify() method and try to find a pattern or a way to navigate to the quotes.

- It is noticed that all the quotes are inside a div container whose id is 'all_quotes'. So, we find that div element (termed as table in above code) using **find()** method :
  ```
  table = soup.find('div', attrs = {'id':'all_quotes'})
  ```
  The first argument is the HTML tag you want to search and second argument is a dictionary type element to specify the additional attributes associated with that tag. **find()** method returns the first matching element. You can try to print **table.prettify()** to get a sense of what this piece of code does.
- Now, in the table element, one can notice that each quote is inside a div container whose class is quote. So, we iterate through each div container whose class is quote.
  Here, we use findAll() method which is similar to find method in terms of arguments but it returns a list of all matching elements. Each quote is now iterated using a variable called **row.**
  Here is one sample row HTML content for better understanding:



  Now consider this piece of code:

```
for row in table.find_all_next('div', attrs = {'class': 'col-6 col-lg-3 text-center margin-30px-bottom sm-margin-30px-top'}):
    quote = {}
    quote['theme'] = row.h5.text
    quote['url'] = row.a['href']
```

```
    quote['img'] = row.img['src']
    quote['lines'] = row.img['alt'].split(" #")[0]
    quote['author'] = row.img['alt'].split(" #")[1]
    quotes.append(quote)
```

We create a dictionary to save all information about a quote. The nested structure can be accessed using dot notation. To access the text inside an HTML element, we use **.text :**

```
quote['theme'] = row.h5.text
```

We can add, remove, modify and access a tag's attributes. This is done by treating the tag as a dictionary:

```
quote['url'] = row.a['href']
```

Lastly, all the quotes are appended to the list called **quotes.**

- Finally, we would like to save all our data in some CSV file.

```
filename = 'inspirational_quotes.csv'
with open(filename, 'w', newline='') as f:
    w = csv.DictWriter(f,['theme','url','img','lines','author'])
    w.writeheader()
    for quote in quotes:
        w.writerow(quote)
```

Here we create a CSV file called inspirational_quotes.csv and save all the quotes in it for any further use.

So, this was a simple example of how to create a web scraper in Python. From here, you can try to scrap any other website of your choice. In case of any queries, post them below in comments section.

2. Demonstrate scrapy framework with an exmaple to strore infromation about an product having attributes: name, cost, manufactorer, rating on to a database (Consider: SQLITE). URl: www.shopping-TV.com. here the website is not allowed to scarpe using bots or script. (TLO: 1.2) (6M)

Need to Describe about Scrapy Project Structure namely:

1. Spiders file (1)
2. Items and commands to execute (1)
3. Pipelines (2)
4. Middleware (1)
5. Settings (1)

3. Write a regular expression for matching URL under following considereation:
    1. Must start with http or https or ftp followed by ://
    2. Must match a valid domain name
    3. Could contain a port specification (http://www.sitepoint.com:80)
    4. Could contain digit, letter, dots, hyphens, forward slashes, multiple times (TLO: 1.1) (4M)

Regular expression:

```
^(http|https|ftp):[\/]{2}([a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,4})(:[0-9]+)?\/?([a-zA-Z0-9\-\._\?\,\'\/\\\+&amp;%\$#\=~]*)
```

4. Write a program to create 3x4 array, delete 3rd column and insert new column in the 3rd column. Print the intermediate results. (TLO: 2.1) (4M)

```
import numpy

print("Printing Original array")
sampleArray = numpy.array([[34,43,73],[82,22,12],[53,94,66]])
print (sampleArray)

print("Array after deleting column 2 on axis 1")
sampleArray = numpy.delete(sampleArray , 1, axis = 1)
print (sampleArray)

arr = numpy.array([[10,10,10]])

print("Array after inserting column 2 on axis 1")
sampleArray = numpy.insert(sampleArray , 1, arr, axis = 1)
print (sampleArray)
```