This is just a very basic intro to OOP. Will be helpful in using various libraries.
*Difference between OOP and Structured Programming:
In structured programming languages, the problem is broken down into smaller problems and implemented in functions. OOP creates object to do the same. An object may refer to anything - Customer, car, cake…
In OOP the functions (methods) and variables (fields) that have a logical connection are put together in classes. For eg- Functions and variables that handle people in a game can be put in a class called People.

*Difference between class and object:
Class contains definition for the object. Its like having a recipe but unless you make the cake, the recipe is of no use. You can have a class that defines a car, what all can the car do, what should be its properties… unless instantiated its of no use.

Example-

```
class Car:
    tyres = 4
my_car = Car()
Print(my_car.tyres) #Prints 4
```

*Class v/s Instance variables:
Class variables belong to the class. Instance variables belong to instances of the class. Instance variables can only be defined within methods. Class variables outside any methods.
Example-

```
class Car:
    def noTyres(self,n):
        self.tyres = n
my_car1 = Car()
my_car1.noTyres(4)
my_car2 = Car()
my_car2.noTyres(3)
Print("my_car1: "my_car1.tyres+"my_car2: "+my_car2.tyres)
#Prints my_car1: 4 my_car2: 3
```

Self refers to the new created object. Its just a variable that "points" to the created object. Not a keyword but just a standard. For example when my_car1 was created and methods were called for my_car1, self referred to it and did the same when we called methods on my_car2.

Just a small intro to some OOP concepts mainly encapsulation, abstraction and inheritance.

*Encapsulation:
Its restricting access to certain methods and variables from outside the class.
Example-

```
class Car:
    def __init__(self):
        self.__stereo_volume = 45 #Can only be accessed from #another method
    def adjustVol(self,level): #Setter method
        self.__stereo_volume = level
    def showVol(self): #Getter method
        print(self.__stereo_volume)
mycar = Car()
mycar.adjustVol(50)
mycar.showVol() #50
mycar.__stereo_volume = 80 #Doesn't do anything
mycar.showVol() #50
```

Same with methods.

*Inheritance
A class can inherit methods and attributes from another class, called a base/super class. Python supports multiple inheritance.

Example-

```
class Movies:
    def directorName(self,name):
        self.director = name
    def duration(self,time):
        self.time = time
    def showDetails(self):
        print("Director: "+self.director+" Duration: "+str(self.time))

class Interstellar(Movies):
    pass
new_movie = Interstellar()
new_movie.directorName("Christopher Nolan")
new_movie.duration(2.45)
new_movie.showDetails()
```

Example-

```
class Movies:
    def directorName(self,name):
        self.director = name
    def duration(self,time):
        self.time = time

class Interstellar(Movies):
    def directorName(self,name): #Overiding
        self.director = "Director: "+name
```

```
    def showDetails(self):
        print(self.director+" Duration: "+str(self.time))
new_movie = Interstellar()
new_movie.directorName("Christopher Nolan")
new_movie.duration(2.45)
new_movie.showDetails()
```

*Constructors:
Python uses __init__ as constructor. This method is called just after the creation of an object. Constructors allow passing of arguments during instantiation and is helpful in initialising variables.

Example:

```
class Person:
    def __init__(self,name):
        self.name = name
    def sayHello(self):
        print("Hey! I am "+self.name+".")

new_person = Person("Hariharan")
new_person.sayHello()
```

*IMDB Webscraper
Webscraping, simply put, is just gathering information from the web. Say you have a list of movies with about 5000 of them and you want to get the rating, plot summary and poster for each movie. Its nearly impossible to go to each movie's page and gather that information manually. This is where automation comes in and Python is at the for front when it comes to scripting and automation. We will be building something like this.

Requests:
HTTP protocol is based on request and response. The client (Could be a browser, program or anything that can send a request) sends a request to the server that responds with a response. There are formats that as to be followed but we won't be looking into it. We will use a library called 'requests' in Python to send and receive HTTP requests. There are mainly 2 requests we should now about: POST and GET. There are others like PUT, DELETE but we don't need them.

Post:
Is used to submit data to a particular resource. Eg: Forms.

```
r = requests.post('http://example.com', data = {'name_of_the_input':'value'})
```

Get:
Is used to get data from a particular resource. Can also be used to pass parameters along with the URL.

```
r = requests.get('http://example.com')

----------------------------------------------------------------------------------

payload = {'key1': 'value1', 'key2': ['value2', 'value3']}

r = requests.get('http://example.com', params=payload)
```

print(r.url) #http://example.com
print(r.status_code) #200 is connection established correctly
print(r.text) #Returns the source code (String, utf encoded)
r.content #Returns binary data

Inspect the web page:

Imdb-Scraper:

imdb_scraper.py

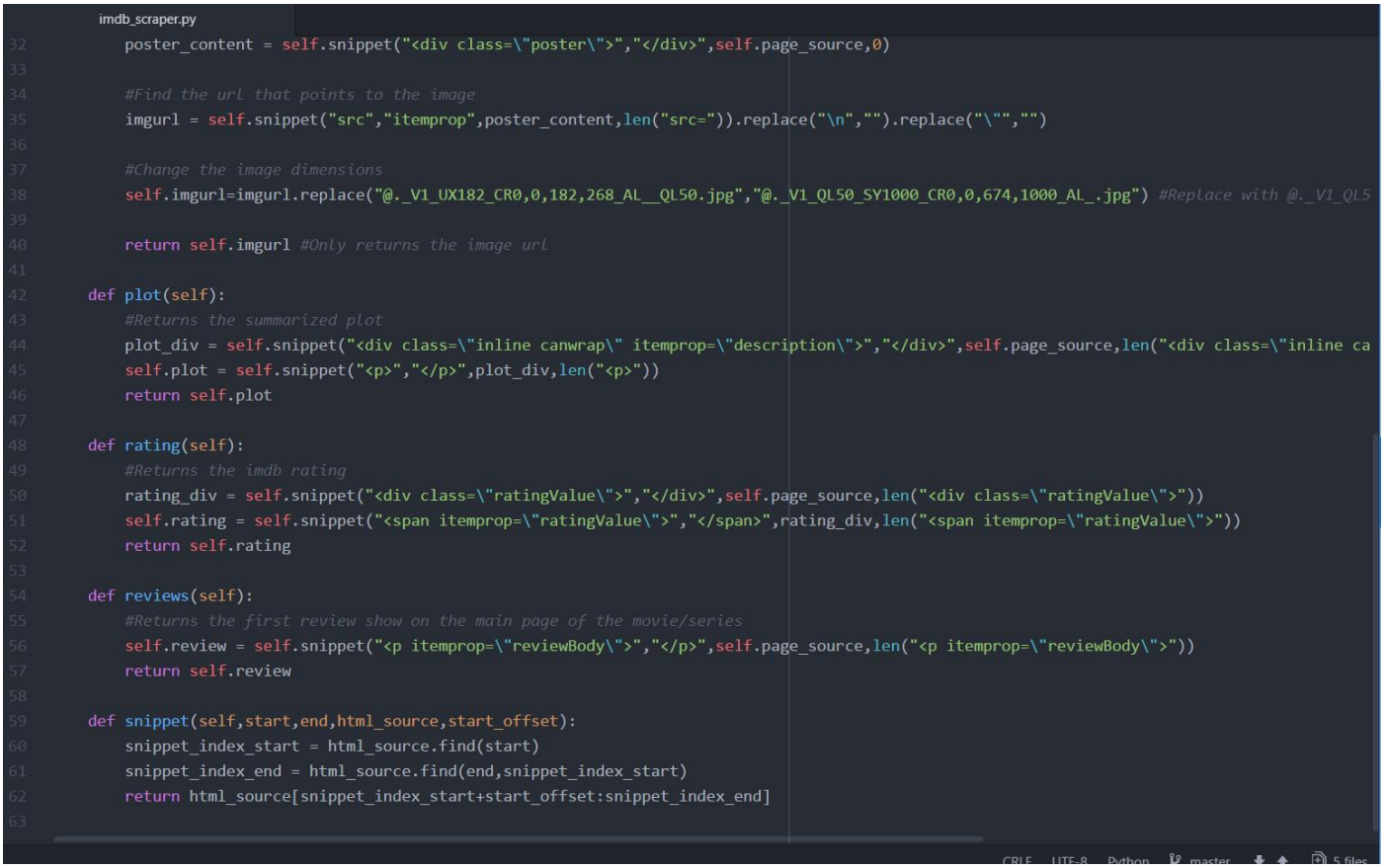
```
import requests,sys
class Scraper:
    def __init__(self,search,toSearch):
        if toSearch:
            self.search_source = requests.get("http://imdb.com/find", params={'q':search}).text
            first_result = self.snippet("Titles</h3>","</a>",self.search_source,len("<h3 class=\"findSectionHeader\"><a name=\"tt\"></a>Titles<
            url = self.snippet("<a href=",">",first_result,len("<a href="))
            self.url = "http://imdb.com"+url.replace("\"","") #Remove double-quotes
            print(self.url)
            self.page_source = requests.get(self.url).text
            #print(url)
        else:
            self.url = search
            html_source = requests.get(self.url).text
            self.page_source = html_source

    def title(self):
        #Returns the title of the movie
        title_div = self.snippet("<div class=\"title_wrapper\">","</span>",self.page_source,len("<div class=\"title_wrapper\">"))
        self.title = self.snippet("<h1 itemprop=\"name\" class=\"\">"," ",title_div,len("<h1 itemprop=\"name\" class=\"\">"))
        return self.title

    def movie_id(self):
        #Returns the imdb id using the url
        self.movie_id = self.url.split("/")[4]
        return self.movie_id

    def poster(self):
        imdb_url = self.url

        #Find the poster division and isolate it from the rest of the html source
        poster content = self.snippet("<div class=\"poster\">","</div>",self.page source,0)
```



Using the library:

```
import imdb_scraper,requests
imdb =
imdb_scraper.Scraper("http://www.imdb.com/title/tt4116284/?ref_=nv_sr_5",False)
plot = imdb.plot()
rating = imdb.rating()
title = imdb.title()
imgurl = imdb.poster()
movie_id = imdb.movie_id()
#Save the poster
image = open(movie_id+".jpg","wb")
image_data = requests.get(imgurl).content
image.write(image_data)
image.close()
review = imdb.reviews()
print("Title: "+title+"\n"+"Rating: "+rating+"\n"+"Plot: "+plot+"\n"+"Poster:
"+imgurl+"\n"+"Review: "+review)
```