Week 6 – Python

1. **Function part 1:**
   Definition:
   Functions are the way to achieve modularity and reusability in Code.
   - *Modularity* comes from module, we create different modules in which we have different function and when we need that function, we call it from that module.
   - *Reusability* means if there is a one task that is frequently performed, we can add that in the function and when we need it, we can call the function instead of writing it from the scratch.

2. **Function Part 2:**
   Defining the Function:
   To define the function, we use keyword **def** with **function name, pair of square brackets** and **colon sign**. Then from the indented line **body of function** will be started. **Brackets are the representation of Function.**

   Proper Syntax and Sample Code:

```
def add():
    a = int(input("Enter first number: "))
    b = int(input("Enter Second number: "))
    print(a+b)
```

   We named this function as **add** after colon we have body of function that will be executed when it is called. In this case when we will call this function it will ask the user for the input of two number and will print their sum.
   With this much code it will not do anything.

3. **Function Part 3:**
   Calling the Function:
   We cannot get any result from the function until and unless we will call it. Interpreter will not go in the body of function till we call that function. As soon as we call function interpreter will start executing body of the function.

   Syntax to call the function:
   We have to write **function_name** and **pair of square brackets.**

   Code:

```
def add():
    a = int(input("Enter first number: "))
    b = int(input("Enter Second number: "))
    print(a+b)

add()
```

The function will be executed and it will give following result:
Enter first number: 12
Enter Second number: 12
24

It asked for two numbers and give sum, that is what we asked in function. Now we can use it several times.

4. **Function Part 4:**
Parameters of Function:
Parameters of function are the variable passed in the definition of function. We gave these variables in the square brackets after the name of function.
Code:

```
def add(a,b):
    print(a+b)
```

Now we don’t need to write input statement to get values.

Arguments of Function:
We declared the variables so, we have to give them some value too. Arguments of function are values passed to the variables in function call. These are also known as the positional arguments. Because they assign value to variable according to position i.e. First variable will have first value and second variable will have second value.
Code:

```
def add(a,b):
    print(a+b)
add(10,15)
```

It will assign 10 to a and 15 to b. Like in the sequence. **Result will be:**
25

If we change the values passed result will also be changed.

Code:

```
def name(first_name , last_name):
    print(first_name+last_name)
name("Usman","Yaqoob")
```

So, here Usman will be assigned to first_name and Yaqoob to last_name.
Result:
UsmanYaqoob

But if we change the values:

```
def name(first_name , last_name):
    print(first_name+last_name)
name("Yaqoob","Usman")
```

Result:
YaqoobUsman

5. **Function Part 5:**
Keyword Arguments:
As we seen in positional arguments, in this type of passing values/arguments we use variable names in arguments to pass the value so even sequence got disturb, it does not disturb the result.
Code:

```
def add(a,b):
    print(a+b)
add(b=10,a=15)
```

So, in above code we made the values opposite but it will not affects anything because 10 will assign to b and 15 to b.
Result:
25

Code:

```
def name(first_name , last_name):
    print(first_name+last_name)
name(last_name="Yaqoob",first_name="Usman")
```

Result:

UsmanYaqoob

We can also mix positional and keyword but keyword will come in the end.

```
def name(first_name , last_name):
    print(first_name+last_name)
name("Usman", last_name="Yaqoob")
```

UsmanYaqoob

6. **Function Part 6:**

Default Parameters:

Default parameters are the parameters that have the default value provided by the user in definition of function so that if user do not give arguments in the function call it chooses that value.

Code:

```
def name(first_name ,last_name,middle_name = " "):
    print(first_name+middle_name+last_name)
name("Usman", "Yaqoob")
```

As we can see in above code person don't have the middle name so we gave space to middle name as a default parameter. Default parameters should be in last.

Result will be:

Usman Yaqoob

OR:

```
def name(first,middle,last=" Khan "):
    print(first+middle+last)
name("Usman"," Yaqoob")
```

Its First name and middle name is given in arguments but last name will be picked from the default arguments.

So, Result will be:

Usman Yaqoob Khan

It can get clearer with the following line of code:

```
def sum(number2,number1=0):
    print(number1 + number2)
sum(number2= 5)
```

We gave default value of number1 variable and so we did not give its value in arguments or function call so it will use the default value given in the parameters.

Result will be:

5

If we change the value of default argument in function call then the latest value will be ap[pear in the result which will be that we passed in the function call.

Code:

```
def name(first,middle,last="  "):
    print(first+middle+last)
name("Usman"," Yaqoob"," Khan")
```

Result will be:

Usman Yaqoob Khan

7. **Function Part 7:**
   Dealing with an unknown number of arguments:
   In some functions we exactly do not know that how many numbers of arguments user would pass while calling the function so we need a parameter that can take all the provided by the user.
   One parameter can store more than one value. We use " * " symbol for that kind of parameter. It will deal with arbitrary number.

   Code and working can be understood by following lines of code:

```
def pizzaorder(size,flavour,*toppings):
    print(f"Your Pizza of size {size}, flavour {flavour} with the toppings of
{toppings} is ready:)")
pizzaorder("large","Chicken Tikka","Olive","Fruits","Chicken")
```

   So, the thing that will happen here is that: we want to order pizza having more than 1 topping, so we used * and made arbitrary parameter. When we will call function and size and flavour will get their values then interpreter will understand that there must be more than 1 values in toppings. So, after the values of flavour all the values will be stored in toppings parameter in the form of tuple.

   Result will be:

   Your Pizza of size large, flavour Chicken Tikka with the toppings of ('Olive', 'Fruits', 'Chicken') is ready:)

8. **Function Part 8:**
   Passing Information Back from function:
   Function not only performs a given task but it also can return some value which can be assigned, reuse or modified.

   We use keyword **return** for the purpose.

```
def sum(num1,num2):
    ans = num1+num2
    print( ans)
    return ans
result = sum(12,12)
print(result*2)
```

   We stored sum in variable ans and print it and also then we use **return** with ans it will return the value of sum. In next statement we store the function call in result variable. So whenever we will call function, it will return sum which will be stored in the result variable. We can perform more operations on that returned value.

So, its result will be:
24
48

- We can return more then one value or thing. That will be stored in the variable.

```
def sum(num1,num2):
    ans = num1+num2
    print( ans)
    return ans, "Hello World"
result = sum(12,12)
print(result)
```

Result:
24
(24, 'Hello World')

- Don't add anything after return statement because after it because it will directly go to that statement that must be returned so it(statements after return) will not be executed.

9. **Function Part 9:**
   Using Functions as Variables:
   We can use function as variables. This is done by using function call in our expressions and they act like the variables.

   Code:

```
def sum(num1,num2):
    addition = num1+num2
    return addition
def sub(num2,num1):
    subtraction = num2-num1
    return subtraction

result = sum(2,3)+sub(3,2)
print(result)
```

Here we use sum function and sub function as variables in the result expression.
Result:
6

10. Function Part 10:

Local and Global Variables:

- Local variables are variables that are defined inside the functions or we can say that are defined in the body of the function. These variables are not accessible outside the function.
  Code:

```
def happy():
    name = "Saad"
    print(f"{name} is very happy today")
happy()
```

  So, we made a local variable “name” we call the function it will be executed properly.
  Result:
  Saad is very happy today

  But if we try to print the name outside the function. It will give error.

```
def happy():
    name = "Saad"
    print(f"{name} is very happy today")
happy()
print(name)
```

  Result:

```
Traceback (most recent call last):
  File "C:/Users/WAQAR/PycharmProjects/helloworld/app.py", line 5, in <module>
    print(name)
NameError: name 'name' is not defined
Saad is very happy today
```

- Global Variables are those variables that are defined outside the function or outside the body of function. And can be accessed and modified in and outside the function, their scope is Global.
  Code:

```
name = "Saad"
def happy():
    print(f"{name} is very happy today")
happy()
print(name)
```

  In this we defined variable name before function/outside and it will print name.

Result:

Saad is very happy today

Saad

11. Function Part 11:

Function within a Function:

If we make a function, we can also add a function call with in the definition of that new function so that when we will call new function other function will be executed.
It can better understand through sample of **Code:**

```
def commissioncalculator(sales):
    if sales>100:
        return sales*100
    elif sales>50:
        return sales*50
    elif sales>30:
        return sales*30
    else:
        return 0
def salarycalculator(basicsalary,sales):
    grosssalary = basicsalary+commissioncalculator(sales)
    print(f"Your salary with commission is {grosssalary}")
salarycalculator(40000,120)
```

Explanation:

So, firstly we made a function of **commissioncalculator** and then we made a new function **salarycalculator.** We call the commissioncalculator in the definition of salarycalculator function. At this it will go in the definition of commissioncalculator and will get the desired values.

Result:

Your salary with commission is 52000

Another Example:

```
def tax(volts):
    if volts>=100:
        return volts*20
    elif volts>=50:
        return volts*10
    elif volts>=20:
        return volts*5
def bill(bill_without_tax,volts):
    total_bill = bill_without_tax+tax(volts)
    print(f"Your Bill for this month is {total_bill}")
bill(300,100)
```

Result:

Your Bill for this month is 2300

12. Loops:

We have already studied for loops now we will get to know about while loops.

While Loops:

While loops are used for the same purpose as like the for loops but there is simple difference in it. The difference is that it allows user to terminate the loop while setting flags.

Flags:

Flags is just a variable that we use. And we write code that if a specific condition comes in the while loop, this flag become false or true an d terminates the loop.

Syntax of While Loop:

```
a=0
while a<5:
    print(a, "I am While loop")
    a=a+1
```

So, we declared a variable and give it a initial value. After using while loop statement we increment the value a.

Result:
0 I am While loop
1 I am While loop
2 I am While loop
3 I am While loop
4 I am While loop

Why we use while loop:

```
a=0
while a<5:
    userinput= input("Enter your favorite football player")
    a=a+1
```

If you see the code it will ask me the same question 5 times. What if I have only 2 favorite players or I have more than 5 favorite players. So, while loop gives ability to user to terminate the loop before condition met by creating one more condition.

This can be understood by sample of Code:

```python
flag = True
favoriteplayer=[]
while flag:
    userinput= input("Enter your favorite football player ")
    if userinput =="Terminate":
        flag = False
    else:
        favoriteplayer.append(userinput)

print(f"Your Favorite players are {favoriteplayer}")
```

So, in this I gave the True value to flag variable in the very start. And use it after while, when it will become false loop will be terminated. Now when ever you will put Terminate or any word you declare loop will terminated.

Result:

```
Enter your favorite football player Ronaldo

Enter your favorite football player Messi

Enter your favorite football player Terminate

Your Favorite players are ['Ronaldo', 'Messi']
```

13. Classes:

Python is object-oriented programming language. This means that almost all the code is implemented using a special construct called classes.

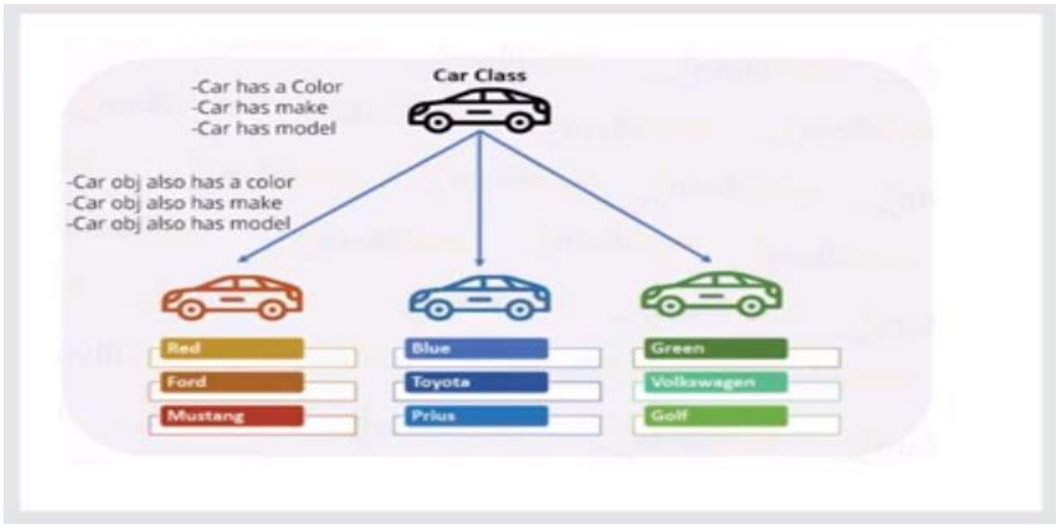
What is class:

Class is the base of object-oriented programming. Programmers use classes to keep related things together. This is done using the keyword "class".

By definition class is a **model**, **blueprint(map), template of something.** When we say template, it is clear that we can make other things like that class.
A class may also be defined as something that can be followed to create **objects** and **instances**.

When we will create objects of the class, those objects will follow the class. They will be similar to the class.

Simple example of class and object:



Explanation:
Every car must have a specific color, make (company) and model. So, when the objects of car will be created, they also must have these. So, they will be following the class. For object of class it is necessary that object must have all the things of that class.

Defining a Class:
For defining a class, we use key word **class** which is followed by **class name** with **square brackets()** and we end the line with **colon.**
After definition **body of class is started with indentation.**
Example

```
class Car():
    #body of class
```

What class holds:
First thing that the class have is **variables** or we can say **attributes.**
For example, if car is a class then car must have color, make and model etc. These all are the attributes that the class will have.

Second thing class holds is **functions** or we can say **behaviors**.
For example, if car is a class, a car can move, stop, speed up, speed down. These all are the behaviors or we can say functions.

Creating objects or instances:
In python almost everything is object, with its properties and methods. We make object to access the variables and functions of the class. If an object belongs to a class it must have been following the requirements set by the class.

To create an object, write **name of object**, **assignment operator** which is = and then **name of class with brackets.**
Example

```
class car():
    #body of class
Car1 = car()
Car2 = car()
```

Explanation through Complete Example:
Defining and making variables:

```
#define a class
class Car():

#making variables/attributes
def __init__(self, make, model, year):
    self.make = make
    self.model = model
    self.year = year
    #default attribute
    self.battery = "2000amp"
```

So, here for creating variables we will use a special function which is **__init__().** We will give all variables as parameters of function. **Self** is first variable, we can use any name for it but we will always firstly use it. We use self because it makes copies of variable and function for every object of class.

Defining Functions:

```
#defining functions/behaviors
def desciptioncar(self):
    print(f"The make of car is {self.make}")
    print(f"The model of car is {self.model}")
    print(f"The year of car is {self.year}")
def move(self):
    print(f"{self.make} is moving with the speed")
def apply_break(self):
    print(f"{self.make} has applied breaks")
def stop(self):
    print(f"{self.model} has been stopped")
def showbattery(self):
    print(f"{self.make} has battery of {self.battery}")
def setbatterySize(self,newSize):
    self.battery = newSize
def getbatterySize(self):
    print(f"The size of your car battery is {self.battery}")
```

These all are the functions or the behavior that a class have. Through setbatterySize function we can change the battery size. There is another way of changing values of attribute but this is more appropriate.

Creating Objects:

```
#creating objects of the class
car1 = Car("Honda", "Civic", 2020)
car2 = Car("Toyota", "Mehran", 2010 )
```

We have made the objects and we gave the arguments (values to parameters). There ar two objects but we can make as much as we like.

Using Objects:

```
#Getting info from class using object
print(car1.make)
print(car2.make)
print(car1.year)
print(car2.model)
```

We use object name , dot and attribute that we want. So, it will print:

Honda

Toyota

2020

Mehran

Similarly, we also can get info from ***functions***.

```
#Getting info from class using object
car1.move()
car1.apply_break()
car1.stop()
car2.stop()
car2.showbattery()
```

Honda is moving with the speed

Honda has applied breaks

Civic has been stopped

Mehran has been stopped

Toyota has battery of 2000amp

Changing value of an attribute:

```
#Changing an attribute value
#Direct hit
car1.battery = "300amp"
print(car1.battery)
```

Now, we have changed the default value of battery for car1 now if we print it, it will show:

300amp

```
#changing Through function
def setbatterySize(self,newSize):
    self.battery = newSize
def getbatterySize(self):
    print(f"The size of your car battery is {self.battery}")
```

Second method that we can use is through function which is better and to get values instead of printing statement we can use function too.

```
car1.setbatterySize("3000amp")
car1.getbatterySize()
```

It will print:

3000amp

The size of your car battery is 3000amp

Complete Code:

```python
#define a class
class Car():
    #making variables/attributes
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        #default attribute
        self.battery = "2000amp"

    #defining functions/behaviours
    def desciptioncar(self):
        print(f"The make of car is {self.make}")
        print(f"The model of car is {self.model}")
        print(f"The year of car is {self.year}")
    def move(self):
        print(f"{self.make} is moving with the speed")
    def apply_break(self):
        print(f"{self.make} has applied breaks")
    def stop(self):
        print(f"{self.model} has been stopped")
    def showbattery(self):
        print(f"{self.make} has battery of {self.battery}")
    #changing Through function
    def setbatterySize(self,newSize):
        self.battery = newSize
    def getbatterySize(self):
        print(f"The size of your car battery is {self.battery}")
#creating objects of the class
car1 = Car("Honda", "Civic", 2020)
car2 = Car("Toyota", "Mehran", 2010 )

#Getting info from class using object
car1.move()
car1.apply_break()
car1.stop()
car2.stop()
car2.showbattery()
#Changing an attribute value
#Direct hit
car1.battery = "300amp"
print(car1.battery)

car1.setbatterySize("3000amp")
car1.getbatterySize()
```

My practice:

```python
class student():
    def __init__(self, name, roll_no, result,apercentage):
        self.name = name
        self.roll_no = roll_no
        self.result = result
        self.apercentage = apercentage
        self.behaviour = "Good"
    def basic_info(self):
        print(f"{self.name} is student of our school")
        print(f"Student's Roll no is {self.roll_no}")
        print(f"He {self.result} the latest exam")
        print(f"{self.name} is {self.apercentage} today")
        print(f"{self.name} have {self.behaviour} behaviour in class")
    def attendance(self):
        print(f"{self.name} is {self.apercentage} today")
    def behaior(self,behav):
        self.behaviour = behav
        print(behav)
    def getbehavior(self):
        print(self.behaviour)

student1 = student("Usman Yaqoob", "1", "Top", "present")
student1.getbehavior()
student1.behaior("Very very  very GOOOOOOOOOOOOOOOOOOODdddddd")
```

14. Data Files:

Unlike word processing files we lose all the variables, dictionaries, lists and classes etc. in python ones we turned off our PC. Through data files we can save data processed by python. It enables us to write data in a data file and also, we can read data from data files.

We use "**with open**" function for accessing and creating data files. It requires two parameters one is file name that we want to access and other is mode.

Syntax:

with open("file_name.txt","mode")

Files Modes:

There are 3 basic modes which are:

I. *Writing to a text File:*
With this mode we can write in text file and data in it. If we write in a file that does not exist. Writing mode will create that file and will add that data in it.

Syntax:

```
With open(“file_name.txt”,“w”) as file:
          File.write(“Data you want to add”)
```

When this function will run it will return a handler which does not have a name so we use **as** to give it a name. We give it name of file in above but we can use whatever we want.

Code:

```
with open("myfile.txt", "w") as usman:
    usman.write("Hey i am usman Yaqoob")
```

It will create a file and will write this in it.

II. *Reading from a text file:*

It enables us to read data from the file. It does not read data if file does not exist and will it give error when it is executed.

Syntax:

```
 With open(“file_name.txt”,“r”) as file:
          Content = file.read()
Print(content)
```

We declare a variable and stored data in it so that we can easily print our data.

Code:

```
with open("myfile.txt","r") as usman:
    content = usman.read()
print(content)
```

Result:

Hey i am usman Yaqoob

III. Append Mode:

There is one problem is writing mode that it we use write mode on file that already contain some data, it will delete all the old data and it will then write new data in it.

Append mode is solution of this issue. If we use append mode all the old data will be saved and it will start writing new data after old one.

Syntax:
with open(“file_name.txt”,“a”) as file:
    file.write(“data you want to enter”)

Code:

```
with open("myfile.txt","a") as file:
    file.write(". I am 18 years old.")
```

Now if we read that file:

```
with open("myfile.txt","r") as file:
    c = file.read()
print(c)
```

It will print:

Hey i am usman Yaqoob. I am 18 years old.

There are two more modes that are derived from the read and write mode which are:

IV. *w+ Mode:*

This mode actually allows read and write both in the files. But if file is not created and we try to use this mode it will create that file and will read from it.

Syntax:
with open(“file_name.txt”,“w+”) as file:
    file.write(“Data you want”)
    print(file.read())

Code:

```
with open("newFile.txt","w+") as file:
    file.write("This is new file")
    file.seek(0)
    print(file.read())
```

We use seek function because if we don’t use it, it will print nothing because it first writes the data to the last index and it is present at last index so we have to take it back to first word like 0 index.

Result:
This is new file

V. r+ Mode:

Similar to the w+ mode it also allows us to read and write the data in file only if it is created. If file is not created and we try to write in it, it will give error.

Syntax:

```
with open(“file_name.txt”, “r+”) as file:
        file.write(“Data you want”)
        print(file.read())
```

Code:

```
with open("newFile.txt","r+") as file:
    file.write("Here is r+ mode demonstration")
    file.seek(0)
    print(file.read())
```

It will print:

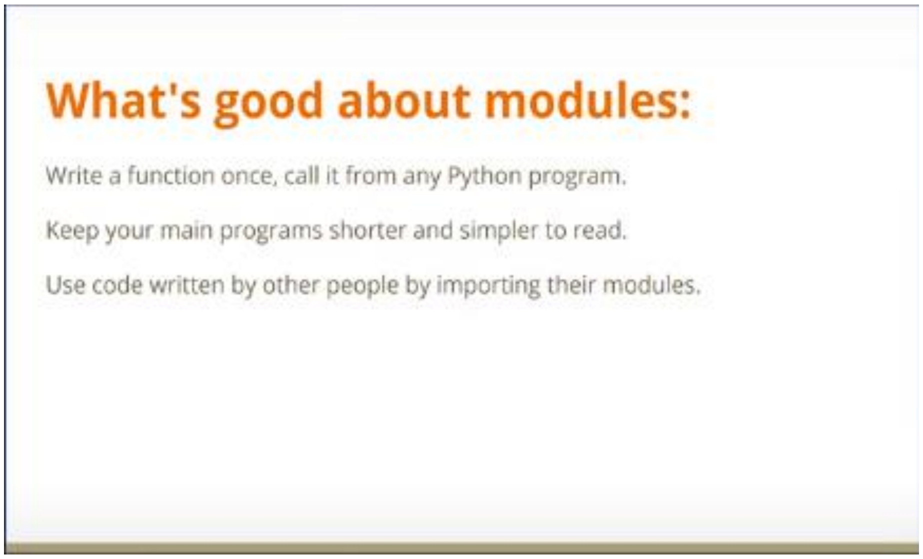
Here is r+ mode demonstration

15. Modules:

Python enables us to store functions in a separate file to reduce the size of class or program. So, when we need it we can use it by importing the whole file by just using word **import.** This file is known as Module.

Syntax of Importing:

Import (Name of module)



What's good about modules:

Write a function once, call it from any Python program.

Keep your main programs shorter and simpler to read.

Use code written by other people by importing their modules.

First make a module File:
Code:

```
def sum(a,b):
    print (a+b)

def sub(a,b):
   print(a-b)

def mul(a,b):
    print(a*b)
```

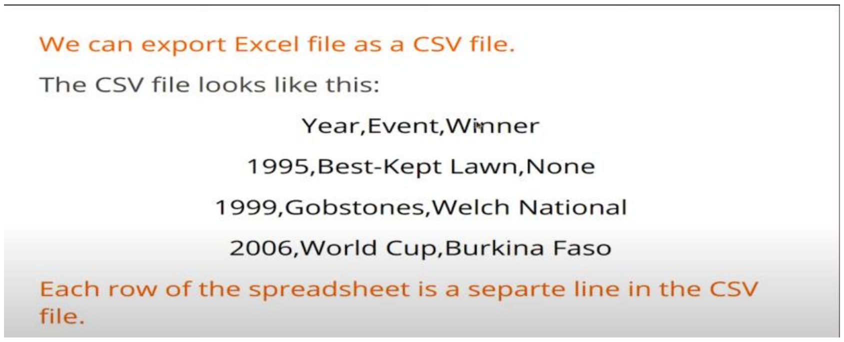
We saved it as “maths”.
Now we have to import it by using the syntax:

```
import maths
maths.sum(1,2)
maths.mul(3,3)
maths.sub(3,2)
```

Result:
3
9
1

16. CSV Files:

CSV stands for *Comma-Separated Values*. CSV files are text only files that are simplified versions of a spreadsheet or database. Every value is separated by a comma.
Like Data Files we can read, write and append CSV files too. WE can use excel sheet as a CSV file that has rows and columns.



We can export Excel file as a CSV file.

The CSV file looks like this:

```
Year,Event,Winner
1995,Best-Kept Lawn,None
1999,Gobstones,Welch National
2006,World Cup,Burkina Faso
```

Each row of the spreadsheet is a separte line in the CSV file.

Note that Every row of spreadsheet will create a sperate line in the CSV file.

I. ***Reading a CSV File:***
We can read CSV file like data file and the same limitation stays which is we can not read the file that is not created.

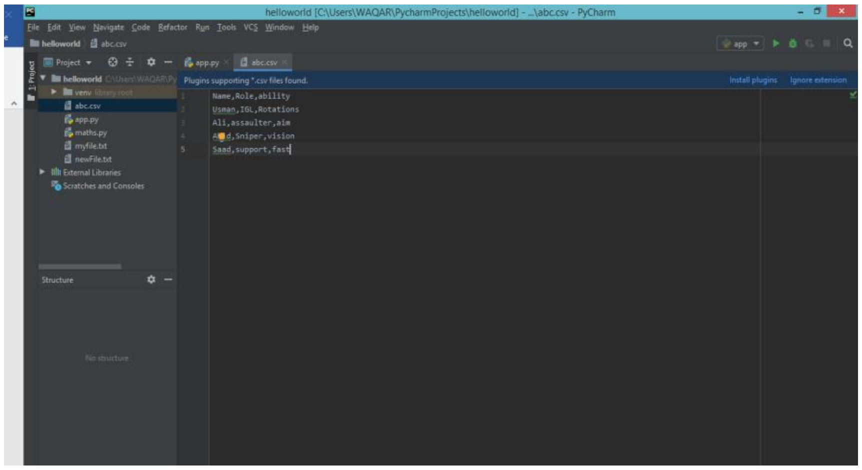
Syntax:

```
import csv
with open("file_name.csv") as f:
        content = csv.reader(f)
```

We do not use read or write parameter because if we do not give that parameter automatically it means that we want to read file.
The content of the CSV file returned by the csv.reader function are not useable yet or they will not be visible in the output. We have to loop through the data stored in the content line by line, adding each line to a list.
For that Syntax will be:

```
import csv
with open("file_name.csv") as f:
        content = csv.reader(f)
        content_list = []
        for each_line in content:
                content_list+= each_line
```

We created an empty list and then extract a line one by one and added it into that list.
First, we have to make a csv file and then we will perform function on it.


helloworld [C:\Users\WAQAR\PycharmProjects\helloworld] - ...\abc.csv - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
helloworld
abc.csv
Project
app.py
abc.csv
maths.py
myfile.txt
newFile.txt
venv library root
External Libraries
Scratches and Consoles
Plugins supporting *.csv files found.
Install plugins
Ignore extension
Structure
No structure

```
Name,Role,ability
Usman,IGL,Rotations
Ali,assaulter,aim
A[illegible]d,Sniper,vision
Saad,support,fast
```

So, I have made a csv file now let's read it.
Code:

```
import csv
with open("abc.csv" ) as f:
    content = csv.reader(f)
    for data in content:
        print(data)
```

We do not make any list it will directly give a list.
Result:
['Name', 'Role', 'ability']
['Usman', 'IGL', 'Rotations']
['Ali', 'assaulter', 'aim']
['Abid', 'Sniper', 'vision']
['Saad', 'support', 'fast']

II. *Writing to a CSV File:*

For writing to a CSV file, we have to follow the following *Syntax:*

```
with open(“file_name.csv”, “w”, newline = “”) as file:
        data_handler = csv.writer(file)
        data_handler = csv.writerow([“Name”, “Role”, “ability ”]
        data_handler = csv.writerow([“Usman”, “IGL”, “Rotations ”]
        data_handler = csv.writerow([“Ali”, “assaulter”, “aim ”]
```

So, in first line we gave new line parameter and assign it to no space because we want to add data continuously in front. Delimeter is space, comma or full stop anything in between two words. So, here we gave comma. And through write row function we added the rows in the csv file.
If we write to file that contain data it will be overwrite.

Code:

```
import csv
with open("xyz.csv", "w", newline="") as file:
    file_writer =  csv.writer(file)
    file_writer.writerow(["Name", "Surname", "Nationality" ])
    file_writer.writerow(["Saad", "Saleem", "Pakistani"])
    file_writer.writerow(["Usman", "Yaqoob", "Pakistani"])
    file_writer.writerow(["Huzaifa", "Khalid", "Pakistani"])
    file_writer.writerow(["Hassaan", "Shahid", "Pakistani"])

with open("xyz.csv") as file:
    content = csv.reader(file)
    for data in content:
        print(data)
```

It will make a file of name “xyz” and will read it.
Result:
['Name', 'Surname', 'Nationality']
['Saad', 'Saleem', 'Pakistani']
['Usman', 'Yaqoob', 'Pakistani']
['Huzaifa', 'Khalid', 'Pakistani']
['Hassaan', 'Shahid', 'Pakistani']

III. Appending to a CSV File:

Similar to data files if we write something to an existing file its old csv data will be removed and new will be added. So, in CSV files too we have a feature to add something in an existing file through appending.

Syntax:

```
with open(“file_name.csv”, “a”, newline = “”) as file:
        data_handler = csv.writer(file)
        data_handler = csv.writerow([“new feature”, “new”, “new ”]
```

Code:

```python
import csv
with open("xyz.csv", "w", newline="") as file:
    file_writer =  csv.writer(file)
    file_writer.writerow(["Name", "Surname", "Nationality" ])
    file_writer.writerow(["Saad", "Saleem", "Pakistani"])
    file_writer.writerow(["Usman", "Yaqoob", "Pakistani"])
    file_writer.writerow(["Huzaifa", "Khalid", "Pakistani"])
    file_writer.writerow(["Hassaan", "Shahid", "Pakistani"])

with open("xyz.csv", "a", newline= "") as file:
    file_writer = csv.writer(file)
    file_writer.writerow(["Danial", "Waheed", "Pakistani"])
    file_writer.writerow(["Furqan", "Shah", "Pakistani"])

with open("xyz.csv") as file:
    content = csv.reader(file)
    for data in content:
        print(data)
```

Those two lines will be added in it and *Result will be:*

```
['Name', 'Surname', 'Nationality']
['Saad', 'Saleem', 'Pakistani']
['Usman', 'Yaqoob', 'Pakistani']
['Huzaifa', 'Khalid', 'Pakistani']
['Hassaan', 'Shahid', 'Pakistani']
['Danial', 'Waheed', 'Pakistani']
['Furqan', 'Shah', 'Pakistani']
```

17. JSON File:

It stands for *Java Script Object Notation.* Like both of files we studied earlier json also has function of reading, writing and appending.

i. *Writing list and dictionary to JSON File:*

For writing we use word ***dump.***

Syntax:

import json

name of list/dictionary = [{list / dictionary to be written}]

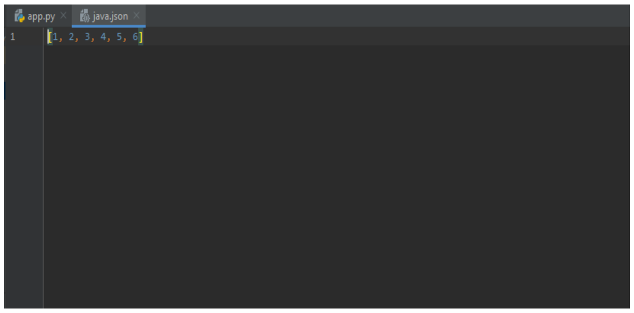
with open(“file_name.json”, “w”) as file:

        json.dump(name of list/dictionary, file)

Code:

```
import json
list = [1,2,3,4,5,6]
with open("java.json","w") as file:
    json.dump(list, file)
```

So, it will make a json file that will contain this list.



```
[1, 2, 3, 4, 5, 6]
```

ii. *Reading Data from JSON File:*

For reading data from a json file we use word *load.*

Syntax:

import json

with open(“file_name.json”, “r”) as file:

        content = json.load(file_name,file)

        print(content)

Code:

```
import json
list = [1,2,3,4,5,6]
with open("java.json","w") as file:
    json.dump(list,file)

with open("java.json", "r") as file:
    content = json.load(file)
    print(content)
```

Result:
[1, 2, 3, 4, 5, 6]

iii. Appending Data in a JSON File:

Syntax:

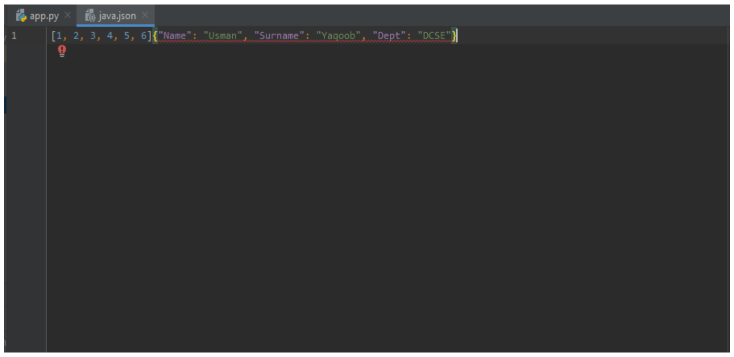
import json
name of list/dictionary = [{list / dictionary to be appended}]
with open("file_name.json", "a") as file:
json.dump(name of list/dictionary, file)

Code:

```
import json
dictionary = {"Name" : "Usman",
              "Surname" : "Yaqoob",
              "Dept" : "DCSE"}

with open("java.json", "a") as file:
    json.dump(dictionary, file)
```

Result:



```
[1, 2, 3, 4, 5, 6]{"Name": "Usman", "Surname": "Yaqoob", "Dept": "DCSE"}
```

18. Exceptions:

Exceptions are run time errors. These errors are not from the developer's side but happen due to the user. Like when he enters wrong value. These errors are also known as *Run time errors.*

Exception Objects:

Python uses special objects called exceptions to manage errors that arise during a program's execution.
Whenever an error occurs that makes Python unsure what to do next, it creates an exception object.

Handling Exceptions/Error:

In Python two keywords are used to handle the exceptions which are *try* and *except.*

A try - except block asks Python to do something, but it also tells Python what to do if an exception is raised.
If you use try – except blocks, your programs will continue running even if things start to go wrong. One try block might have more than one exception block.

Finally:
Finally, is a block which is executed/run in every condition even if exception happens or not.

Code:

```
a = int(input("Enter a : "))
b = int(input("Enter b :"))
try:
    a/b
except ZeroDivisionError:
    print("Exception Caught")
else:
    print("No exception happens")
finally:
    print("I will always be here")
```

So, when we will run this, *First result:*
Enter a : 4
Enter b :4
No exception happens
I will always be here

Let's change values:
Enter a : 5
Enter b :0
Exception Caught
I will always be here