tut6(comment and escape sequence)

```
print("hello")
# single line comment
"""multi line
comment
"""

print("hello avi")
print("hello avi2")
# Now every print statement is starting in new line to end it in same line use
 end="(character with which you want to end like comma,space,fullstop)"
print("hello avi",end=", ")
print("hello avi2")

print("hello avi",end="*****")
print("hello avi2")

print("hello avi","hello avi2",)        #it will automatically give space.
#Escape Sequence - "\n,\t"  to print escpae sequence write \\ like "\\n ,\\t "
```

tut7 (variables)

```
var1="hello world"
var2 =4
var3 =36.7
var4 = "Avi"
print(type(var1))
print(type(var2))
print(type(var3))

#print(var1 + var2)  error
#print(var1 + var3)  error
print(var2+ var3)
print(var1+var4)
print(type(var2+ var3))


#Type casting int(variable_name)         int(),float(),str()
var5 = "4"
var6 = "12"
print(var5+var6)
print(int(var5)+int(var6))
print(str(var2)+str(var3))

#utilities  10*(any_string) print it 10 times, n*(any string) print it n times
```

```
#how to take input in python input is taken by default as string
print("Enter no.")
a = input()
print("You entered ",a)
#alternate method
a = input("enter again")
print("You entered ",a)
```

tut8 (string slicing)

```
mystr ="Avi is good boy"
print(mystr) #print complete string
print(mystr[2]) #print of that index.
print(len(mystr)) #print the length
print(mystr[0:10]) #print from including starting index and upto last index(ex
cluding)
print(mystr[0:10:2]) #print with escaping one characters if instead of 2 we wr
ite n it will skip n-1 character
print(mystr[::-1]) #first reverse the string then print from starting to last
print(mystr[::-
2]) #first reverse the string then print from starting to last skiping 1 chara
cter
print(mystr[0:])  #print from starting index to last
print(mystr[::]) #print from starting to last

print(mystr.isalnum()); #check whether it is alpha numeric or not
print(mystr.isalpha()); #check whether it is alpha or not here false b/c conta
ins space
print(mystr.endswith("boy")) #check whether it ends with given word/letter or
not
print(mystr.count("o")) #counts no of repetition
print(mystr.capitalize()) #capitalise first letter
print(mystr.find("is")) #gives the starting index
print(mystr.lower()) #convert in lowercase
print(mystr.upper()) #convert in uppercase
print(mystr.replace("good","very good")) # replace
```

tut9 (list,tuple and list func.)

```
grocery =["harpic","vim bar","dio","bhindi","lollly pop", 45]
print(grocery) #print complete list
print(grocery[3]) #print at that index assume as array
numbers =[5,5,5,8,3,7,4]
numbers.sort() #sort
numbers.reverse() #reverse
numbers.append(9) #append
```

```
numbers.insert(1,61) #insert 61 at index 1
numbers.remove(5)
numbers.pop() #remove last element
print(numbers.count(6)) #count how many times it is appeared in list
#slicing
#slicing returns new list but not changes intial list
print(numbers[:]) # it will print the list with index starting before colon(by
 default 0) and just before index mentioned after colon(by default last index)
#extended slicing
print(numbers[::2]) # print the list skipping 1 element if n then skipping n-
1 letter if -1 then reverse list
print(len(numbers))  #find length
print(max(numbers)) #find maximum in list
print(min(numbers)) #find minimum in list

"""List are mutable while tuple is immutable means tuple can not be changed,
 tuples are in parenthesis while list is in square bracket """
tuple_example =(4,6,5)
print(tuple_example)
"""to make a tuple of single element put a comma after the element like (1,)""
"

"""simple program to swap two numbers"""
a=1
b=2
a,b = b,a
print(a,b)

####################### LIST COMPREHENSION ##########################
"""
A list comprehension generally consist of these parts :
   Output expression,
   input sequence,
   a variable representing member of input sequence and
   an optional predicate part.

For example : list for square of odd no

lst  =  [x ** 2  for x in range (1, 11)   if  x % 2 == 1]

here, x ** 2 is output expression,
      range (1, 11)  is input sequence,
      x is variable and
      if x % 2 == 1 is predicate part.
"""
```

tut10 (Dictionary)

```
#Dictionary is key value pair it is written in {}
```

```
d1 ={"Avi":"Avishek","Abh":"Abhay","Rau":"Raushan"}
print(type(d1))
print(d1["Avi"]) #it will print the value of corresponding to that key
# we can also keep dictionary inside dictionary
d2 ={"Avi":{"Name":"Avishek","MOB":7479734685},"Abh":"Abhay","Rau":"Raushan"}
print(d2["Avi"]["Name"])
d2["Ankit"] = "Junk Food" # add items in dict
print([d2["Ankit"]])
d2["Ankit"] = "New Junk Food"
print([d2["Ankit"]])
del d2["Ankit"] # it will delete this key and its pair
d3 = d2  # it means now d3 will point d2 , on changing in d3 it will change on
 d2
d3 = d2.copy() #it will return copy of d2, now d2 and d3 will be independent

print(d2.keys())  #print keys
print(d2.items()) #print items
```

tut11 (Set)

```
s = set()
print(type(s))
s_fron_list = set([1,2,3,4])  # same set can be created using list instead of
passing list you can also pass the name of list declared before
print((s_fron_list))
s.add(1)  # it will add it in set but set contains unique values only
s.add(1)  # adding it again will not change any thing in set but in list you c
an add repeatitively
print(s)
s1 = s.union({4,2,3})   #return union of s and given set
print( s,s1)
s1 = s.intersection((1,3)) # return union of s and given set
print( s,s1)
print(s.isdisjoint(s1)) #it will tell whether s ar s1 is disjoint or not
s_fron_list.remove(4)   # removes the passed element
#similarly len(s),max(s),min(s) will gives length and maximum, minimum value
```

tut12 and 13(if else)

```
a = 6
b= 56
c = int(input("Enter no\n"))

if c>b:
    print("Greater")
elif c==b:
    print("Equal")
```

```
else:
    print("Lesser")


list1 = [5,7,3]
if 5 in list1:
    print("Yes")
if 15 not in list1:  # here "in" and "not in" is keyword
    print("NO")

#short hand for if else
c = int(input("Enter no\n"))
if c>b : print("Greater")
c = int(input("Enter no again\n"))
print("Greater") if c>b else print("Smaller")
```

tut16 17(for and while loop)

```
list1 = ["avi","kvi","ashi","aksd","klajdkf","sdfkld"]
list2 = [["akldj",1],["klaj",2],["klaj",3]]
dict1 = dict(list2)
for item in list1:
    print(item)
for item,numb in list2:
    print(item,"has scored rank",numb,"\n",end="*")
print("printing dictionary ",dict1)
#to iterate in dictionary
for item,numb in dict1.items():
    print(item,"has scored rank",numb)

#practise
prac = ["avi",4,12,"akd",6.5]
for item in  prac:
    if(str(item).isnumeric() and item>6):
        print(item)
    else:
        print("It is not allowed")

"""range function it will take three argument max,
first is starting, second is terminating and third is increment value
if you pass one value then by default starting value =0, increment value = 1,t
erminating value = passed value
if you pass 2 value (a,b) a= starting value ,b= terminating value,increment va
lue=1(bydefault)
"""
for item in range(0,6,2):
    print(item,end=" fOr ")
print("")
```

```
i=0
while(i<=5):
    print(i,end=" w ")
    i += 1
```

tut 18 (break, continue)

```
i=0
while(i<=10):
    if(i==4 or i==8):
        i += 1
        continue
    print(i, end=" ")
    i += 1
i=0
print("")
while(i<=10):
    if(i==8):
        i += 1
        break
    print(i, end=" ")
    i += 1
```

tut21(Operators)

```
"""Arithmetic,Comparision,Identity,Membership,Bitwise
#Arithmetic
+  addition
-  subtraction
*  multiplication
/  divide
// divide but gives integer GIF
** exponetial
% modulu for remainder

#Assignment Operator
x +=7  means x = x+7

#Comparison Operator
== equality
!= not equal
>,< greater,lesser
>=,<= greater than,less than

#Logical Operator
a = True
b = False
"and" and
"or"  or
```

```
#Identity operator
"is" (a is b) check whether a is b
"is not" (a is not b) check whether a is not b

#Membership Operator
list = [3,3,2,45,24,8,6,7,15,76]
"in" print(45 in list) it will print whether 45 is in list or not
"not in"

#Bitwise Operator
& binary and
| binary or


"""
```

tut 22 (function and docstring)

```
# user define function
#first line inside a function as a multi line comment is called docstring whic
h should contain some details related to that function
#to check the docstring of function f1() write "print(f1.__doc__)"
def function(a, b):
    """this is a function to add two number"""
    print("Hello inside function", a + b)
def function2(a,b):
    """this to find average"""
    average = (a+b)/2
    print(average)
    return average

print(function(7, 10))
print(function2(4,8)) # first execute the function then print the return value
print(function2.__doc__) #print the docstring
print(function2.__code__)
print(function.__doc__)
```

tut23 (exception handling)

```
# Here try and except is used

a = input("enter a\n")
b = input("enter b\n")
try:
    print("sum",int(a)+int(b))
except Exception as e:
    print("something is wrong",e) #Exception is the error msg
```

File handling(reading)

```
"""
FILE IO BASICS
"r" - open file for reading
"w" - open file for writing , erase the previous content and write ,create new
 file if not exist
"x" - creates file if not exist (exclusive creation)
"b" - open for binary mode
"t" - text mode  (it is default mode)
"a" - add more content (append) write in last of file
"+" - both read and write


"""
#f = open("avi.txt")                        #syntax of file opening
f = open("avi.txt","r")                  #second argument is file opening mode
by default it "r" mode
# content = f.read()                          #if we not mention any no. insid
e read() it will read whole file
# print(content)

#content = f.read(3)         #here it will read first 3 character and then cal
ling it again it will read after that character
                            # and if we put very large no such that there is n
ot that much character then it will read upto last only
#print(content)

#content = f.read(3)          #here it will read 3 character after that which
was already read
#print(content)
""" to read file line by line or letter by letter we have to iterate the file"
""
# for line in content:                #it will read letter by letter here we h
ave to use content
#     print(line,end=(" "))
# for line in f:                           # read line by line
#     print(line,end="")              # here no need to use content = f.read()
 b/c if we use it will read file upto end and nothing will left to read

# print(f.readline())    # it will also read line
# print(f.readline())    #when it is called again it will read after that

print(f.readlines())    #it will store the line in list
```

```
f.close()                                   #syntax of file closing and it is
good practise to close the file
```

file handling(writing and appending)

```
# f = open("avi2.txt","w") # erase and write
# f = open("avi2.txt","a")  # append
f = open("avi2.txt","r+")   # handle read and write
print(f.read())
f.write("Thank you")
# a = f.write("Writing in a file again") #it will returns no. of character wri
tten

f.close()
```

tell and seek

tell – returns the position of pointer in terms of character

seek- it changes the position of pointer to given position

```
f = open("avi.txt")
f.seek(11)
print(f.tell())
print(f.readline())
# print(f.tell())

print(f.readline())
# print(f.tell())
f.close()
```

using with block

it is same as f = open and f.close()

```
with open("avi.txt") as f:  #inside open you can also pass opening mode
    a = f.readlines()
    print(a)
```

inside this with block you can do your work related to file and on coming outside of this block it automatically close the file.

Scope, global variable and keyword

```
#first part

l = 10 #Globlal
def function1(n):
     #l = 5          #it is local scope

    # print(l)
    """here it will first search for local variable and if it will not in loca
l scope it will search in global
     we can not change global variable directly it will give error"""
    global l           # need to write this statement to change the the value
of global variable
    l = l + 10
    print(n,l,"I have printed")


function1("This is me")
print(l)


#second part
x=89
def avi():
    x =20
    def avi2():
        global x     # on writing this it will go directly to global scope to
find
        x=88        # here on writing it will directly go to global scope(note
 x = 20 is not in global scope it is also inside a fn) and if there is any var
iable it will change its value or
        # if not create it in global scope and assign it value
    print("befor calling avi2()",x)         # here it will print 20 because it
 accessing its local variable
    avi2()
    print("befor calling avi()",x)      # here it will print 20 again because
it that fn changed in global scope but avi() fn has x in its local scope so it
 will execute this one.
avi()
print(x)  #it will print 88 because that fn has changed your value
```

lambda or anonymus function

it is a one liner function without any name

```
# Lambda functions or anonymous functions
# def add(a, b):
#     return a+b
#
# # minus = lambda x, y: x-y   #both this minus or minus fn defined below do same
#
# def minus(x, y):
#     return x-y
#
# print(minus(9, 4))



a =[[1, 14], [5, 6], [8,23]]
a.sort(key=lambda x:x[1])
print(a)
```

sort or sorted function can receive two arguments " key = none " and "reverse = False" none and false is by default if we not pass if we make reverse = true it will sort in descending order

and key if we pass any function name then it pass each element of that list or tuple in function and then sort according to the returned value of function

and instead of passing a fn you can also write lambda function

using modules

to install a module open windows powershell and run it as administrator then write "pip install flask" to install flask for other write its name

```
import random
import datetime
import playsound
import platform
import math
import builtins
import calendar

print(datetime.datetime.now())
print("Here is the calendar",calendar.month(2020,2))
print(platform.system())
print(math.sqrt(25))
print(builtins.bin(8))
playsound.playsound('Tera zikr.mp3')
```

```
random_num = random.randint(0,5)
print(random_num)
print(random.random())  # 0 to 1
lst = ["channel1","channel2","channel3","channel4","channel5","channel6","chan
nel7"]
print(random.choice(lst))
```

F-string and string formatting

F string is similar to template literal of ES6 of javascript

```
me = "Avi"
a1 =3
# a = "this is %s %s"%(me, a1)
# a = "This is {1} {0}"
# b = a.format(me, a1)
# print(b)
a = f"this is {me} {a1} {math.cos(65)}"   # this is f string
# time
print(a)
```

```
a = input("Enter you name\n")
b = "18"
str = "Hello {0} you have to wait {1} minutes to meet with Mr. Abhishek Pratap
 Singh"
print("Hello %s you have to wait %s minutes to meet with Mr. Abhishek Pratap S
ingh"%(a,b))
print(str.format(a,b))
print(f"Hello {a} you have to wait {b} minutes to meet with Mr. Abhishek Prata
p Singh")
```

*args and **kwargs

args means arguments

kwargs means keyworded (named or like key, pair) arguments

keyworded arguments doesn't matter order in which they passed in function, in python it knows the name of variable name defined(search keyworded arguments in python on google for more info)

The special syntax **args* in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

The special syntax ***kwargs* in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name *kwargs* with the double star. The reason is because the double star allows us to pass through keyword arguments (and any number of them).

```
# def function_name_print(a, b, c, d, e):
#     print(a, b, c, d, e)

def funargs(normal, *argsrohan, **kwargsbala):
    print(normal)
    for item in argsrohan:
        print(item)
    print("\nNow I would Like to introduce some of our heroes")
    for key, value in kwargsbala.items():
        print(f"{key} is a {value}")



# function_name_print("Harry", "Rohan", "Skillf", "Hammad", "Shivam")

har = ["Harry", "Rohan", "Skillf", "Hammad",
       "Shivam", "The programmer"]
normal = "I am a normal Argument and the students are:"
kw = {"Rohan":"Monitor", "Harry":"Fitness Instructor",
      "The Programmer": "Coordinator", "Shivam":"Cook"}
funargs(normal, *har, **kw)
```

```
#in function *args and ** kwargs are optional even
# if you write it in function and not pass then also  it is ok
# means they can accept 0 items inside it

def fun(normal_variable,*myargs,**mykwargs):
    print("This is normal variable",normal_variable)
    print("Now these are args variable")
    for item in myargs:
        print(item,end="    ")
    print("Now these are kwargs with keyworded variable")
    for key,value in mykwargs.items():
        print(f"{key} is related with {value} using key value pair.")

fun("Avishek",*["Avi","Abhay","Raushan","Varun"],**{"Avi":"7479","Abhay":"8540
","Raus":"****"})
```

Time module

```
import time
initial = time.time()

k = 0
while(k<3):
    print("This is me")
    # time.sleep(2)                    # it stops the program for n no. of se
conds n is number that is passed
    k+=1
print("While loop ran in", time.time() - initial, "Seconds")

initial2 =time.time()
for i in range(3):
    print("This is me")
print("For loop ran in", time.time() - initial2, "Seconds")


localtime = time.asctime(time.localtime(time.time()))
print(localtime)
print(time.time())       # it gives total ticks/seconds from epoch
print(time.localtime(time.time()))      #it converts ticks into tuple of year
month day hour min sec
print(time.asctime(time.localtime(time.time())))        # it converts it into
human readable format
```

Virtual environment

A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated python virtual environments for them. This is one of the most important tools that most of the Python developers use.

Why do we need a virtual environment?

- Imagine a scenario where you are working on two web based python projects and one of them uses a Django 1.9 and the other uses Django 1.10 and so on. In such situations virtual environment can be really useful to maintain dependencies of both the projects.
- Imagine another scenario you have create a program using some functions of current versions of pandas or sklearn libraries and then after some year that libray got updated and that function is removed and when you use the same program it will not behave normally. So we can also keep a copy of current versions of libraries and some other tools also in virtual environment which are used by our program.

Virtual Environment should be used whenever you work on any Python based project. It is generally good to have one new virtual environment for every Python based project you work on. So the dependencies of every project are isolated from the system and each other.

How does a virtual environment work?

We use a module named **virtualenv** which is a tool to create isolated Python environments. virtualenv creates a folder which contains all the necessary executables to use the packages that a Python project would need.

Installing virtualenv

- Open a folder(where you want to create virtual environment) then press shift and right click >Open powershell window here
- After opening it
  ```
  pip install virtualenv
  ```
- Then
  ```
  virtualenv my_name
  ```
- It has created a new virtual environment and it is independent like a new born baby. And you can also check there is folder of `my_name` in the selected folder.
- And if you want to create a virtual environment that is not like new born baby or it should contain all site packages of your system interpreter then use command **virtualenv --system-site-packages my_name2**
- You can activate it by ***opening that folder > Scripts > activate.bat***.
  Or from windows powershell ***.\my_name\Scripts\activate***
- Now your virtual environment has been created. Once the virtual environment is activated, the name of your virtual environment will appear on left side of terminal. This will let you know that the virtual environment is currently active.
- Now you can install your packages using command **pip install its_name**.
- To install a particular version suppose to install Django 1.9 use command
  ```
  pip install Django==1.9
  ```

- However if want to share our program with its dependent package , generally we don't share it with whole packages due to its large size. So we keep a record of packages that is used by our program in our **requirements.txt** file.
- To create the requirements.txt file we use command **pip freeze > requirements.txt**
- Or to install the packages of someone else program keep that requirements.txt file in that folder and use command **pip install -r .\requirements.txt**
- Once you are done with the work, you can deactivate the virtual environment by command **deactivate.**

How to execute your program in new virtual environment already created(in pycharm) ?

- Open pycharm>File>Open
- Choose your virtual environment folder and select/open in pycharm.
- Now you can create any program and simply execute as you do normally.