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()
```

Sr.No.	Modes & Description
1	**r** Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	**rb** Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is th default mode.
3	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	**w** Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file f writing.

6	**wb** Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates new file for writing.
7	**w+** Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exi creates a new file for reading and writing.
8	**wb+** Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file do not exist, creates a new file for reading and writing.
9	**a** Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the appe mode. If the file does not exist, it creates a new file for writing.
10	**ab** Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the f is in the append mode. If the file does not exist, it creates a new file for writing.
11	**a+** Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file ope in the append mode. If the file does not exist, it creates a new file for reading and writing.
12	**ab+** Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exis The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

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.

Enumerators

Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.

```
l1 =["item1","item2","item3","item4","item5","item6","item7","item8"]
for index,item in enumerate(l1):
    print(f"{item} is on index number {index+1}")
```

How import works
And if __name__== '__main__'

When we write import means we are putting the codes of that file in our file we can also import our own file.

__name__ is a built-in variable which evaluates to the name of the current module. Thus it can be used to check whether the current script is being run on its own or being imported somewhere else by combining it with if statement.

If the script is executed in its file then name is main but when imported in some file and executed then its name is its file name .

Sometimes we define some very useful function and do some work in a file and if we import that file then it is again executed there, which is not required we need only those defined function for some other operations. In such case we use **if __name__ ==’__main__’** because when it is imported in some other file then it’s name will be different and the condition becomes false and if it is executed in same file then only condition will be true and it’s code will be executed

File1

```
import file2                          # we can also import files written by us
import file3
import sys
print(sys.path)                      # this tells the hierarchical order of places
where the interpreter searches for any imported
                            # file or package so if it is found in first one then
it import that ones not in other
                            #so it is a very good practice to never keep the name
of a file same as any library like
                            # flask, or pandas or sklearn.

print(file2.a)
file2.printjoke("This is me")
file3.add(10, 5)
print(file3.printhar(", Basanti in kutton k saamne mat naachna"))
```

File2

```
a = 7
def printjoke(str):
    print(f"this function is a joke {str}")

if __name__ == '__main__':
    print("inside file2")
```

File 3

```
def printhar(string):
    return f"Ye string harry ko de de thakur {string}"


def add(num1, num2):
    return num1 + num2 + 5


print("and the name is", __name__)

if __name__ == '__main__':
    print(printhar("Harry1"))
    o = add(4, 6)
    print(o)
```

Join function

It joins the list or tuple or ...

```
lst = ["john", "cena", "Randy", "orton", "Sheamus", "khali", "jinder mahal"]

for item in lst:
    print(item, "and", end=" ")
print("others are wwe superstars")
print()
print(" and ".join(lst), " and others are wwe superstars")
```

here both print statements will do the same thing

Map, filter and reduce

Map – it applies a function over all the elements of a list.

To apply same function over all elements of list we can use for loop or map function

Syntax- **map(function_name, list_name)**

First argument is function name without braces and other is list name. Or instead of that we can also pass lambda function.

It returns a object then we have to type cast in list.

```
def square(x):
    return x*x


def cube(x):
    return x*x*x


numbers = ["3", "34", "64", "420"]
numbers = list(map(int, numbers))
numbers[2] = numbers[2] + 1
print(numbers[2])
num = [2, 3, 5, 6, 76, 3]
sq = list(map(lambda x: x*x, num))  # here map will give each element of num list
to fn which will return its square
print(sq)

func = [square, cube]       # list of function
for i in range(6):
    val = list(map(lambda x: x(i), func))    # here map will give each function
from func list to lambda function and here lambda fn is also recieving a fn and
returning f(i)

    print(val)
```

Filter – It filters a list means it return a object of sublist with elements such that for which that function returns true

```
def is_greater_5(num):
    return num>5


lst = [1,2,3,4,5,6,7,8,9]
print(list(filter(is_greater_5, lst)))
```

Reduce - The **reduce(fun,seq)** function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.This function is defined in “functools” module.

Working :

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
- This process continues till no more elements are left in the container.
- The final returned result is returned and printed on console.

```
from functools import reduce

lis = [1, 3, 5, 6, 2]

# using reduce to compute sum of list
print("The sum of the list elements is : ", end="")
print(reduce(lambda a,b : a+b,lis))

# using reduce to compute maximum element from list
print("The maximum element of the list is : ", end="")
print(reduce(lambda a, b : a if a > b else b, lis))
```

Decorators

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure. Decorators are usually called before the definition of a function you want to decorate.

It takes a function as an argument and we can do something inside a decorator and with that function. To add a decorator use the @ symbol and name of that decorator before the definition of that symbol.

```
def dec1(func1):
    def nowexec():
        print("Executing now")
        func1()
        print("Executed")
    return nowexec


@dec1
def who_is_harry():
    print("Harry is a good boy")

# who_is_harry = dec1(who_is_harry)

who_is_harry()
```

Object Oriented Programming

Class

Like C/C++ same type of class is here,

Here we can add some properties (function or variables) to a particular object without changing in template of class.

```
class Employee:
    no_of_leaves = 8


avi = Employee()
abhi = Employee()

avi.name = "Avishek"
abhi.name = "Abhishek"

print(avi.name)
print(avi.no_of_leaves)
print(abhi.no_of_leaves)
print(Employee.no_of_leaves)

avi.no_of_leaves = 5
print("After changing")
print(avi.no_of_leaves)
print(abhi.no_of_leaves)
print(Employee.no_of_leaves)
```

Output

```
Avishek

8

8

8

After changing

5

8

8
```

To get the details of class or object "**its_name.__dict__**"

Self and __init__()(constructor)

Constructor initialise the object.

When nothing in passed as argument inside the methods (functions) then there is "self" has to be passed, self is calling object and inside the function we access it by using dot operator with self.

__init_() is constructor which takes “self” argument which is used to acess its members.

```
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        # this is a constructor and will be called when
        # object is created and we have to pass these arguments
        # at the time creating object.
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"Name is {self.name} and role is {self.role}"
    # here self is is the object from which it is called


avi = Employee("Avishek", "Not fixed", "Programmer")
abhi = Employee("Abhishek", "Not fixed", "Programmer")
# avi.name = "Avishek"
# avi.role = "Teacher"
# abhi.role = "Student"
# abhi.name = "Abhishek"

print(avi.printdetails())
print(abhi.printdetails())
```

Methods

First of all we need to know what is difference between function and methods ?

Functions defined inside a class is called methods and basic difference is “A method **can operate on the data (instance variables) that is contained by the corresponding class**”.

STATIC METHODS

A static method in python must be created by decorating it with **@staticmethod** in order to let python now that the method should be static. The main characteristics of a static method is that they can be called without instantiating the class. This methods are self contained, meaning that they cannot access any other attribute or call any other method within that class.

You could use a static method when you have a class but you do not need an specific instance in order to access that method. For example if you have a class called Math and you have a method called factorial (calculates the factorial of a number). You probably won’t need an specific instance to call that method so you could use a static method. **It can be used without creating any instance or object or class but other two (class, instance) can not be**.

CLASS METHOD

Methods which have to be created with the decorator **@classmethod**, this methods share a characteristic with the static methods and that is that they can be called without having an instance

of the class. The difference relies on the capability to access other methods and class attributes but no instance attributes .

- A class method is a method which is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class. For example it can modify a class variable that will be applicable to all the instances.

"cls" keyword is used to access class variables and It is same for all the objects of that class

Instance Methods

Simply defined functions inside a class (no keywords are required) are instance methods.

This method can only be called if the class has been instantiated. Once an object of that class has been created the instance method can be called and can access all the attributes of that class through the reserved word self. An instance method is capable of creating, getting and setting new instance attributes and calling other instance, class and static methods **. "self" keyword is used to access the class variables and it is different for different objects of same class.**

Check diff between cls and self

The **difference between** the keywords **self** and **cls** reside only on the method type, on one hand if the created method is an instance method then the reserved word **self** have to be used, on the other hand if the method is a class method is a class method then the keyword **cls** must be used.

```
class Employee:
    no_of_leaves = 8


    def __init__(self, aname, asalary, arole):

        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"Name is {self.name} and role is {self.role}"

    @classmethod
    def change_leaves(cls,new_leaves):
        cls.no_of_leaves = new_leaves

avi = Employee("Avishek", "Not fixed", "Programmer")
abhi = Employee("Abhishek", "Not fixed", "Programmer")
avi.change_leaves(100)
```

```
print(avi.no_of_leaves)
```

Class method as a alternative constructor

We can also use class method as constructor. It can be better explained by following example.

```
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role is
{self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
        # params = string.split("-")
        # print(params)
        # return cls(params[0], params[1], params[2])
        return cls(*string.split("-"))


harry = Employee("Harry", 255, "Instructor")
rohan = Employee("Rohan", 455, "Student")
karan = Employee.from_dash("Karan-480-Student")

print(karan.no_of_leaves)
# rohan.change_leaves(34)
#
# print(harry.no_of_leaves)
```

Single level inheritance

Source code:

```
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role is
{self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
        return cls(*string.split("-"))

    @staticmethod
    def printgood(string):
        print("This is good " + string)


class Programmer(Employee):
    no_of_holiday = 56
    def __init__(self, aname, asalary, arole, languages):
        self.name = aname
        self.salary = asalary
        self.role = arole
        self.languages = languages



    def printprog(self):
        return f"The Programmer's Name is {self.name}. Salary is {self.salary} and
role is {self.role}.The languages are {self.languages}"
```

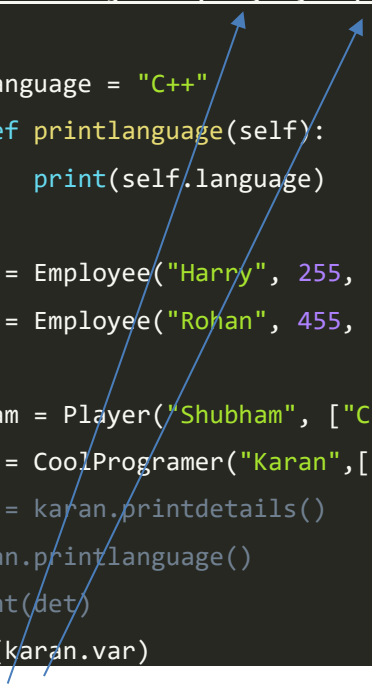
```
harry = Employee("Harry", 255, "Instructor")
rohan = Employee("Rohan", 455, "Student")

shubham = Programmer("Shubham", 555, "Programmer", ["python"])
karan = Programmer("Karan", 777, "Programmer", ["python", "Cpp"])
print(karan.no_of_holiday)
```

Multiple Inheritance

Source Code:

```
class Employee:
    no_of_leaves = 8
    var = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role is
{self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
        return cls(*string.split("-"))

    @staticmethod
    def printgood(string):
        print("This is good " + string)

class Player:
    var = 9
```

```
    no_of_games = 4
    def __init__(self, name, game):
        self.name = name
        self.game =game

    def printdetails(self):
        return f"The Name is {self.name}. Game is {self.game}"

class CoolProgramer(Player, Employee):

    language = "C++"
    def printlanguage(self):
        print(self.language)

harry = Employee("Harry", 255, "Instructor")
rohan = Employee("Rohan", 455, "Student")

shubham = Player("Shubham", ["Cricket"])
karan = CoolProgramer("Karan",["Cricket"])
# det = karan.printdetails()
# karan.printlanguage()
# print(det)
print(karan.var)
```



Order of class matters, if we are using any variable or function that is available in both then it will first search first class passed and if it is there it will use that one.

Multilevel Inheritance

Source Code:

```
class Dad:
    basketball =6

class Son(Dad):
    dance =1
    basketball = 9
    def isdance(self):
        return f"Yes I dance {self.dance} no of times"

class Grandson(Son):
    dance =6
```

```
    guitar = 1

    def isdance(self):
        return f"Jackson yeah!" \
            f"Yes I dance very awesomely {self.dance} no of times"

darry = Dad()
larry = Son()
harry = Grandson()

# print(darry.guitar)
```

In multilevel inheritance if we are using any variable or function which is also present in more than one of its ancesstors then it will use which comes first from that level to grand level.

Public, protected and private

Syntax:

Source Code:

```
class Employee:
    no_of_leaves = 8
    var = 8
    _protec = 9
    __pr = 98

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role is
{self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves
```

```
    @classmethod
    def from_dash(cls, string):
        return cls(*string.split("-"))

    @staticmethod
    def printgood(string):
        print("This is good " + string)

emp = Employee("harry", 343, "Programmer")
print(emp._Employee__pr)
```