Python

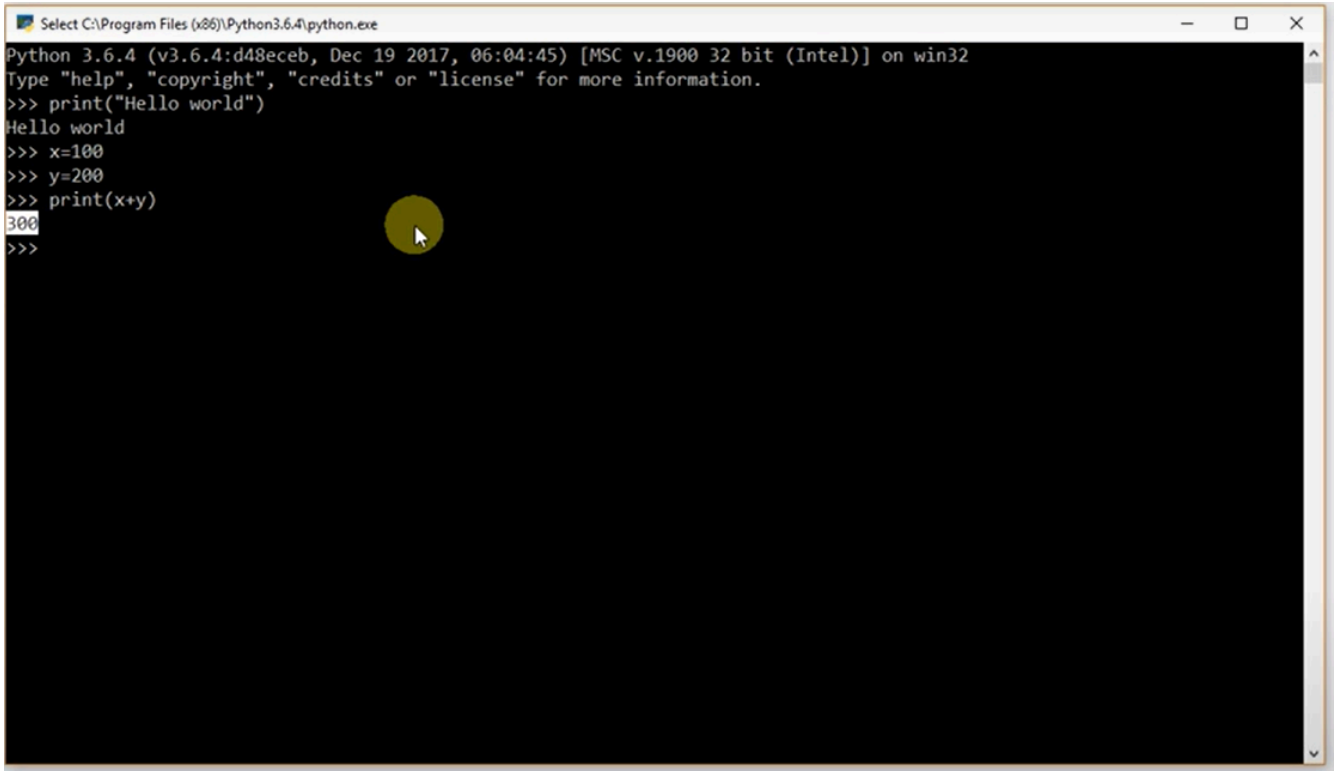
Basics of Python

- Python is High Level, OOP, scripting language
- Platform Independent
- Python is Interpreted language
-

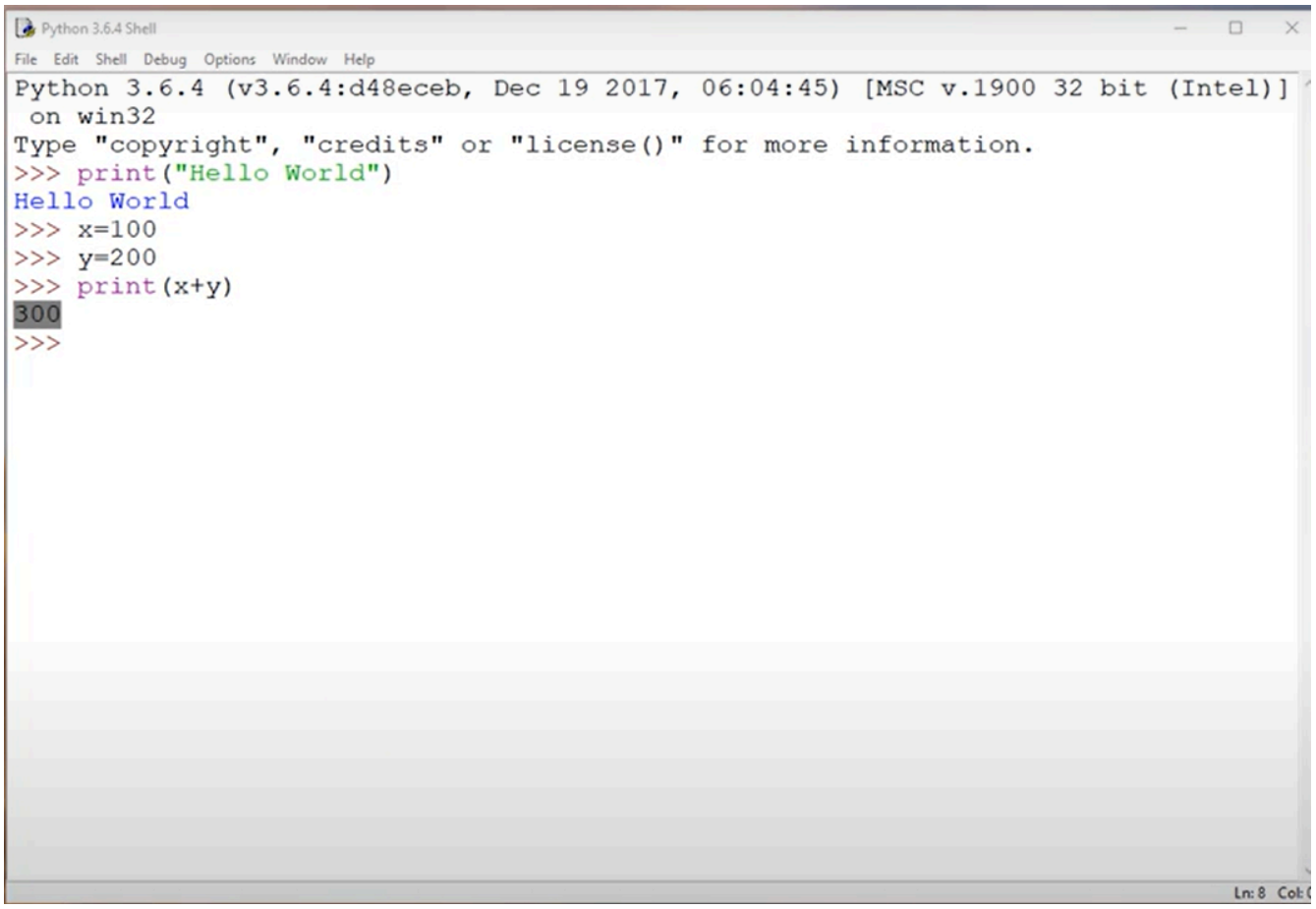
Different Ways to run Python Code -

Add Python Installed directory path in Env Variables.

1. using Python.exe - inside installed directory in windows.
   It gives access to python shell/Terminal. we can write and execute code there.



2. Using IDLE python shell -
   either directly into shell or create new file and then run module.



3. By windows cmd promt -
   write python in windows cmd promt. It will give access to python shell.
   or create a file and then -
   python test.py
4. Online python compiler
5. Using IDE
6. using Eclipse IDE -
   need to install Pydev plugin (by going to help and then install).

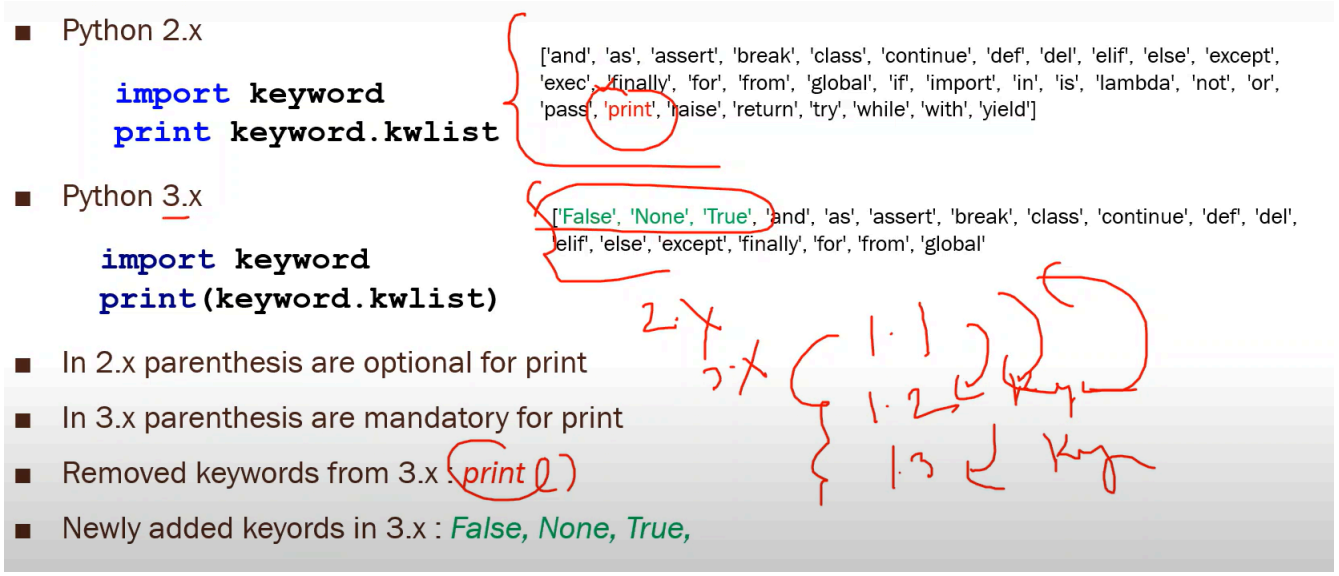
Comments -

- For single line comment, use #
- For multi line, use triple single or double quotes(""" """)
  or (''' ''')

Keywords -

- print is method in python 3 and hence not a keyword.
- Also, parenthesis are mandatory around print in python 3.

- False None , True are newly added keyword in Python 3.



Variables

- No need to mention data type b/c python is dynamically typed
- We can also assign different data types in a single line

```
a=10
b=10.5
name="Pavan"
name1='Kumar'
x=True

print(a)
print(b)
print(name)
print(name1)
print(x)

#this statement assign 100 to a, 10.5 to b and Pavan to name. Multiple Values to multiple varaibles
a,b,name=100,10.5,"pavan"
print(a,b,name)

#a,b,c= 10,10,10
a=b=c=10
print(a,b,c)
```

- In Python we can re-declare the variables
- To delete a variable, `del a`

Data Types Supported

Numbers - int, float

- max() and min() will give max and min out its given arguements.

string
boolean
List
Tuples
Dictionary

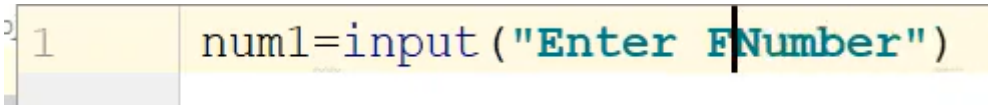
- Use type(x) to get data type of x
- We can concatenate two similar data types only, using + operator
  We can't concatenate string with any other data type.
- We can concatenate numbers and boolean b/c boolean are also 1 and 0 only.
-

Swap 2 variables -

```
x,y = y,x
```

Taking user Input

- In python 3, we have input method, which takes input stores as string.
- Later, we can type cast it to according to our needs
- This is similar to raw_input() in Python 2



- We can't typecast, string containing float to int. Typecasting only possible to same data type as entered by user
- Float can hold int but reverse is not true.

Formatting Output

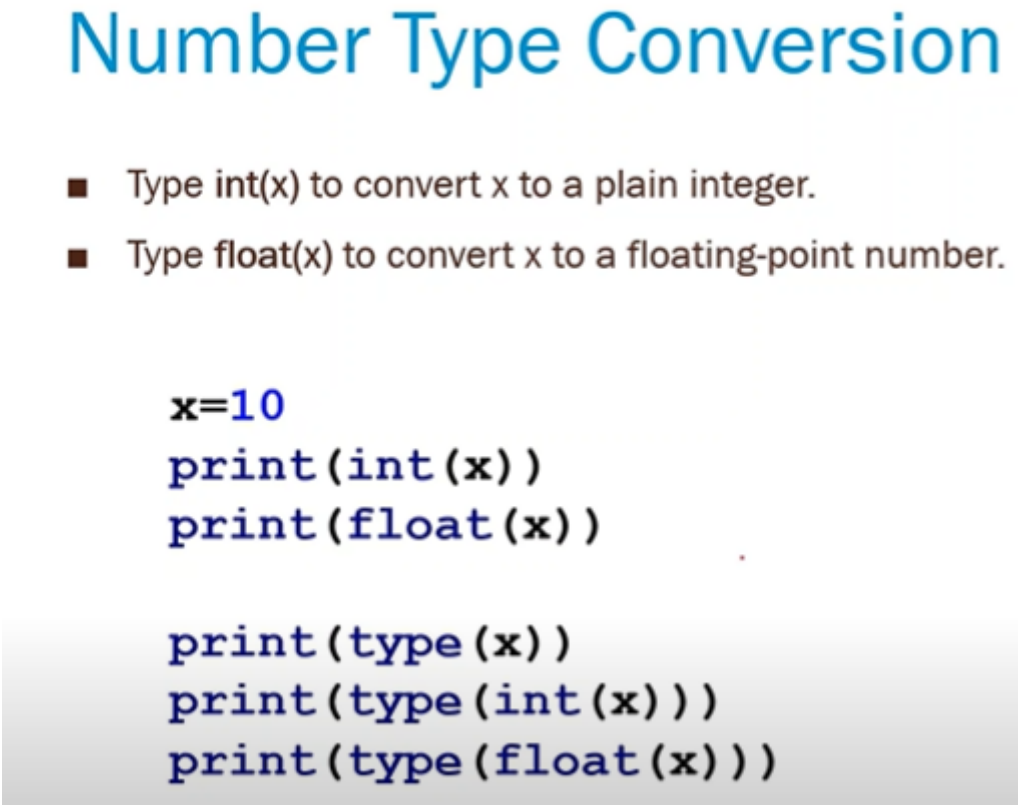
- We can use variable name to print its value
- we can use % opearator.
  print("%d, %s, %g"%(age, name, sal))
- we can use {} operator and .format method.
  print("Name is : {} age is: {} sal is: {}".format(name, age, sal))
- {} along with index
  print("name:{0} age:{1} sal:{2}".format(name, age, sal))

```
name,age,sal = "John",24,10000.35

#Approach1
print(name,age,sal)

#Approach2
print("Name is:", name)
print("Age is:", age)
print("Sal is:", sal)

#Approach3 : using  %  Here type is imp
print("Name:%s  age:%d  salary:%g" %(name, age, sal))

#Approach3 : using  {} Here value is imp
print("Name:{}  age:{}  salary:{}".format(name, age, sal))

#Approach4 : using  {} Here value & index is imp
print("Name:{1}  age:{2}  salary:{2}".format(name, age, sal))
```

Control Statements

1. if else - if and else keyword should be under same line
2. if elif else - here else is optional
3. for loop - for i in range(1,10,2):
   Above loop will start from 1 , go until 10-1 = 9, with inc of 2 at a time.
4. while loop -
   i =10;
   while i>=1:
   print(i)
   i = i-1
5. break - to exit a loop based on certain condition
6. continue - to skip that particular iteration

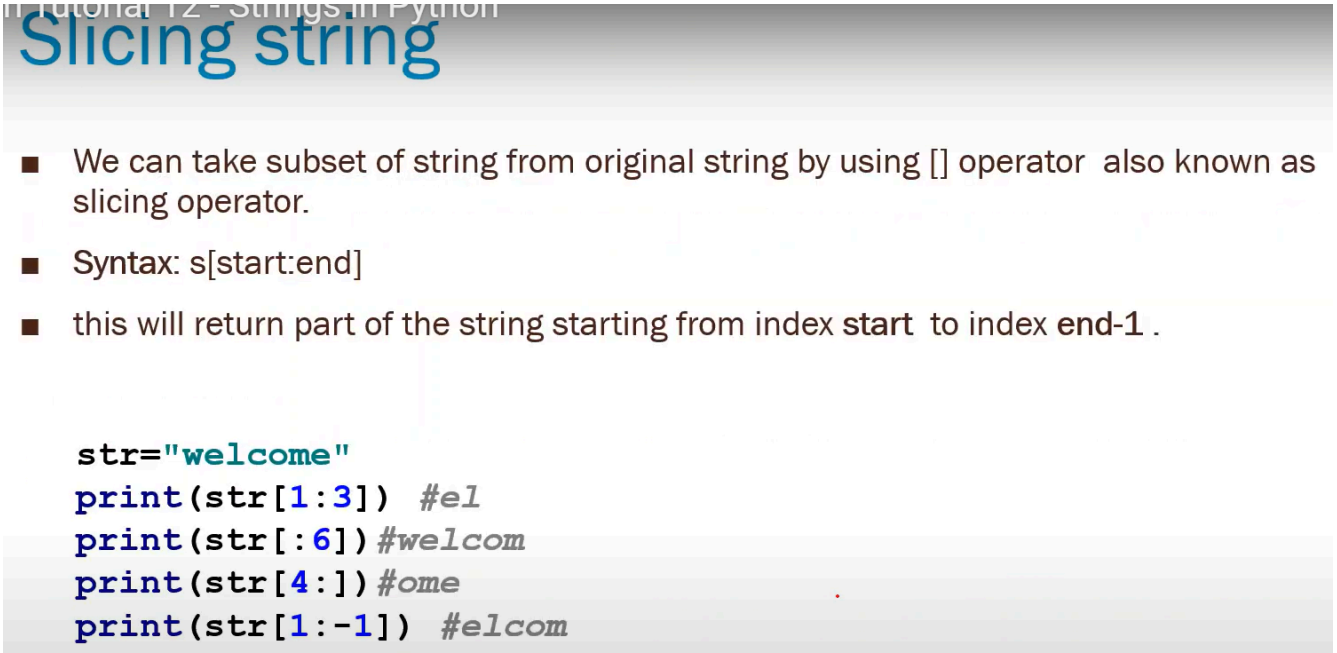
- Number Type Conversion -



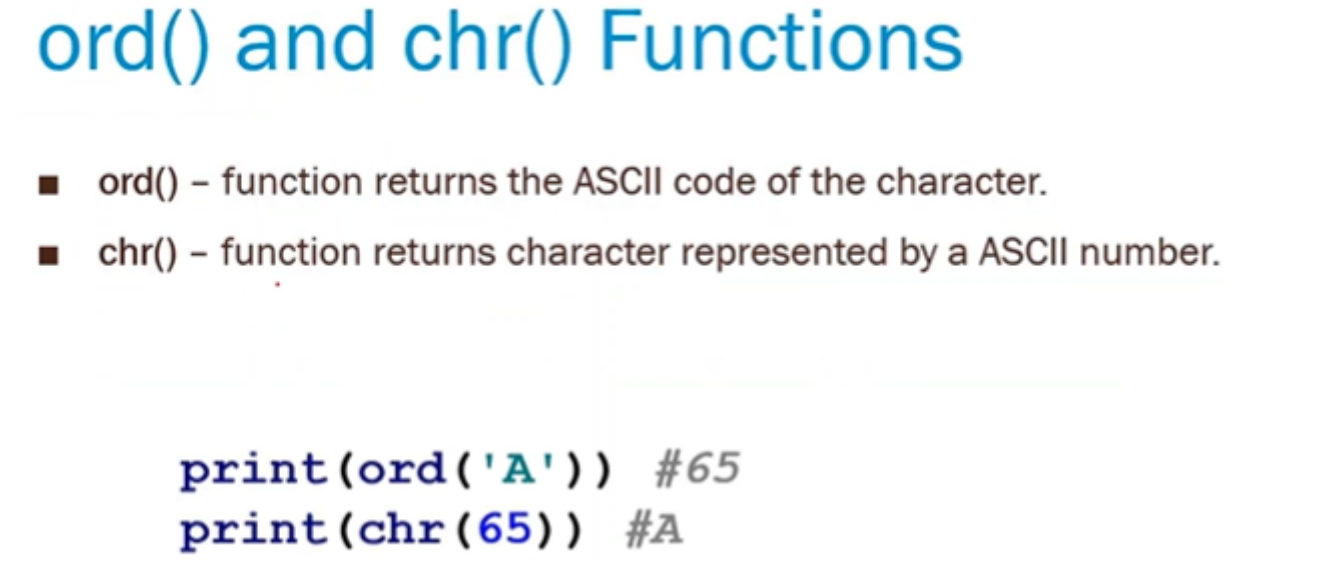
Strings -

- can be declared inside single or double quotes
- typecast user input to string using str(input("Enter a str"))
- str() will create an empty string
  str("John") will add newstring john to str b/c in python re-declarartion is allowed.
- If 2 different var have same content inside it, they will both be on same physical address in memory.
  id(str) will give address of str in memory
- Strings are immutable , i.e. once a str var is created , that same variable can't be modified. If we try to modify that var, then new var with same name at different memory location is created.
- String index starts from 0
- Operations on strings -
  str1+str2 - add two string literal
  str*n - repeat str n times(without adding white spaces)

- Slicing string -



- here , str[1:-1] means it will omit last string character
- ord('A') method returns ASCII value of a character A and chr(65) returns character A corresponding to ASCII code 65



- String Functions -
- len(str), max(str), min(str)

Function name	Function Description
len()	returns length of the string
max()	returns character having highest ASCII value
min()	returns character having lowest ASCII value

```
print(len("hello")) #5
print(max("abc")) #c
print(min("abc")) #a
```

- Membership operators -
  in - returns T if a substring is present inside a string
  not in - returns T if a substring is not inside a string

```
s1 = "Welcome"
print("come" in s1)# True
print("come" not in s1) #False
```

- String Comparisons -
  we can compare 2 strings based on their ASCII characters, matching starts from left to right.

You can use (> , < , <= , <= , == , !=) to compare two strings.

Python compares string lexicographically i.e using ASCII value of the characters.

```
print("tim" == "tie") #False
print("free" != "freedom") #True
print ("arrow" > "aron") #True
print ("right" >= "left") #True
print ("teeth" < "tee") #False
print ("yellow" <= "fellow") #False
print ("abc" > "") #True
```

- ASCII value of empty character is the least
- str is of type sequence , so we can iterate over it.
  - String is a sequence type and also iterable using for loop

```
s = "hello"
for i in s:
    print(i)
    print(s, end="\n")  # this is default behavior
    print(s, end="")  # print string without a newline
    print(s, end="foo")  # now print() will print foo after every
string
```

- Testing Strings -

String class in python has various inbuilt methods which allows to check for different types of strings.

Method name	Method Description
isalnum()	Returns True if string is alphanumeric
isalpha()	Returns True if string contains only alphabets
isdigit()	Returns True if string contains only digits
isidentifier()	Return True is string is valid identifier
islower()	Returns True if string is in lowercase
isupper()	Returns True if string is in uppercase
isspace()	Returns True if string contains only whitespace

```
s = "welcome to python"
print(s.isalnum()) #False
print("Welcome".isalpha()) #True
print("2012".isdigit()) #True
print("first Number".isidentifier())#False
print(s.islower()) #True
print("WELCOME".isupper()) #True
print(" ".isspace()) #True
```

- Search within a string -

Method Name	Methods Description:
endswith(s1: str): bool	Returns True if strings ends with substring s1
startswith(s1: str): bool	Returns True if strings starts with substring s1
count(substring): int	Returns number of occurrences of substring the string
find(s1): int	Returns lowest index from where s1 starts in the string, if string not found returns -1
rfind(s1): int	Returns highest index from where s1 starts in the string, if string not found returns -1

```
s = "welcome to python"
print(s.endswith("thon")) #True
print(s.startswith("good")) #False
print(s.find("come")) #3
print(s.find("become")) #-1
print(s.rfind("o")) #15
print(s.count("o")) #3
```

Converting/Modifying Strings -

- Original string remains the same , we can store modified string in some other str1 var.

Method name	Method Description
capitalize(): str	Returns a copy of this string with only the first character capitalized.
lower(): str	Return string by converting every character to lowercase
upper(): str	Return string by converting every character to uppercase
title(): str	This function return string by capitalizing first letter of every word in the string
swapcase(): str	Return a string in which the lowercase letter is converted to uppercase and uppercase to lowercase
replace(old, new): str	This function returns new string by replacing the occurrence of old string with new string

```
s = "String in PYTHON"
s1 = s.capitalize()
print(s1) #String in python

s2 = s.title()
print(s2)#String In Python

s3 = s.lower()
print(s3) #string in python

s4 = s.upper()
print(s4) #STRING IN PYTHON

s5 = s.swapcase()
print(s5) #sTRING IN python

s6 = s.replace("in", "on")
print(s6) #String on PYTHON

print(s) #String in PYTHON
```

Lists

- List allows to add, delete , process elements easily.
- List is a sequence type and its similar to arrays. But in list we can store same or different data types.
- To create a list -
  list1 = list([22,45,65])
  list2 = list("Python") --> This will create a list of characters, P,y,t,h,o,n

- To access elements of a list

  list[0], list[1]

  List index starts from 0.

- Operations on list -

Method name	Description
x in s	True if element x is in sequence s.
x not in s	if element x is not in sequence s.
s1 + s2	Concatenates two sequences s1 and s2
s * n , n * s	n copies of sequence s concatenated
s[i]	ith element in sequence s.

Method name	Description
len(s)	Length of sequence s, i.e. the number of elements in s.
min(s)	Smallest element in sequence s.
max(s)	Largest element in sequence s.
sum(s)	Sum of all numbers in sequence s.
for loop	Traverses elements from left to right in a for loop.

```
list1 = [2, 3, 4, 1, 32]
print(2 in list1) # True
print(33 not in list1) #True
print(len(list1)) # find the number of elements in the list 5
print(max(list1)) # find the largest element in the list 32
print(min(list1)) # find the smallest element in the list 1
print(sum(list1)) # sum of elements in the list 42
```

- we can also do slicing in list -

  list1[:5] --> takes elements from 0th index to 5-1 =4th index

- To traverse list, use for loop -

  for i in list:

  print(i, end =" ")

- Inbuilt method on Lists -

Method name	Description
append(x:object):None	Adds an element x to the end of the list and returns None.
count(x:object):int	Returns the number of times element x appears in the list.
extend(l:list):None	Appends all the elements in l to the list and returns None.
index(x: object):int	Returns the index of the first occurrence of element x in the list
insert(index: int, x:object): None	Inserts an element x at a given index. Note that the first element in the list has index 0 and returns None..
remove(x:object):None	Removes the first occurrence of element x from the list and returns None
reverse():None	Reverse the list and returns None
sort(): None	Sorts the elements in the list in ascending order and returns None.
pop(i): object	Removes the element at the given position and returns it. The parameter i is optional. If it is not specified, pop() removes and returns the last element in the list.

- List Comphrehension

```
list1 = [ x for x in range(10) ]
print(list1) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

list2 = [ x + 1 for x in range(10) ]
print(list2) #[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

list3 = [ x for x in range(10) if x % 2 == 0 ]
print(list3) #0, 2, 4, 6, 8]
```

Dictionary

- used to store key value pair
- It is a data type
- order is not important in dictionary.
- Dictionaries are mutable
- To create a dictionary -
  friends = {'tom' : '123',
  'jerry' : '456'}
  - Dictionaries can be created using pair of curly braces ({}).
  - Each item in the dictionary consist of key, followed by a colon, which is followed by value.
  - And each item is separated using commas (,).
  - key must be of hashable type, but value can be of any type. Each key in the dictionary must be unique.
- Keys being hashable means all keys should have same data type.
- we can retrieve, add, delete keys in dictionary (i.e. dict are mutable)

```
friends = {'tom' : '111-222-333','jerry' : '666-33-111'}
print(friends) #{'tom': '111-222-333', 'jerry': '666-33-111'}

#Retrieving elements from the dictionary
print(friends['tom']) # 111-222-333

#Adding elements into the dictionary
friends['bob'] = '888-999-666'
print(friends) #{'tom': '111-222-333', 'jerry': '666-33-111', 'bob': '888-999-666'}

#Modify elements into the dictionary
friends['bob'] = '888-999-777'
print(friends) #{'tom': '111-222-333', 'jerry': '666-33-111', 'bob': '888-999-777'}

#Delete element from the dictionary
del friends['bob']
print(friends) #{'tom': '111-222-333', 'jerry': '666-33-111'}
```

  retrieve - friends['key']

add - friends['new key'] = new value
modify - friends['old key'] = new value
delete - del friends['key']

- To traverse a dict -
  for x in friends:
  print(x, ":", friends[x])
- Here, x will become tom, bob ..etc in successive iterations.
- length -- len(friends)
- Two dict are equal if and only if all key and corresponding values are same. we can compare using - == and != operator
- methods in dict class -

Methods	Description
popitem()	Returns randomly select item from dictionary and also remove the selected item.
clear()	Delete everything from dictionary
keys()	Return keys in dictionary as tuples
values()	Return values in dictionary as tuples
get(key)	Return value of key, if key is not found it returns None, instead on throwing KeyError exception
pop(key)	Remove the item from the dictionary, if key is not found KeyError will be thrown

Tuples

- Tuples are like constant arrays. Once declared, we can't add, delete, modify its elements. so, we can say that tuples are immutable.
- To declare a tuple, use ()
  t1 = (1,2,3,4)
  t2 = tuple("abc")
  t3 = tuple([1,3,4,5])
- we can use methods on tuples

```
t1 = (1, 12, 55, 12, 81)

print(min(t1)) #1
print(max(t1)) # 81
print(sum(t1)) #161
print(len(t1)) #5
```

- we can also use for loop to iterate over tuples
  t =(1,4,5,6)
  for i in t:
  print(i, end=" ")

- we can also slice thru tuples
  t = (0,1,2,3,5,6)
  print(t[3:5])
- Also we can use in and not in operator
  print(3 not in t)
  print(2 in t)

List vs Dictionary vs Tuples -

- In dictionary, Duplicate keys are not allowed , value can be duplicated
  Also, we can't do slicing over a dictuonary
- We can't modify Tuples after declarartion

Lists	Dictionaries	Tuples
List=[10,12,14]	Dict={"John":26, "Mary": 30}	Tup1=("10,20,30") or Tup2=10,20,30 Tup3=("John","Scott") or Tup4="John","Scott"
print(List[0])	print(Dict["Mary"])	print(Tup1[0])
Allow duplicates	Duplicates Keys NOT allowed but duplicate values allowed.	Allow duplicates. Faster than Lists.
List[0]=100	Dict["John"]=35	Tuple1[0]=100 *#Type error*
Mutable	Mutable	Immutable – Values can't changed once assingned
[ ]	{ }	( )
Slicing can be done. List=[10,20,30] print(List[1:2]) *#[20]*	Slicing can't be done.	Slicing can be done. tup=(10,20,30,40,50) print(tup[1:4]) *#(20, 30, 40)*
Usage: use lists if you have a collection of data that doesn't need random access. Use lists when you need a simple, iterable collection that is modified frequently.	When you need a logical association between Key:value pair. When you need fast lookup for your data based on a custom key.	Use tuples when your data cannot change. A tuple is used in combination with a dictionary, for example a tuple might represent a key, because its immutable.

Functions

- To create a function -
  def func():
  print("This is a function")
- If you don't want to do anythingh by function, use keyword pass
- Function can return single or Multiple values
  Multi values can be returned using commas and it will be returned as a tuple.
- By default, return is "None". If we dont write anythingh after return , it will return None
- Global var is declared outside function and is accessible anywhere in the script, but local variabes are declared and accessed inside a function only.
- Also, we can declare a global var inside a function itself(as global var) and then it will be accesssible to outside function also.
- Also, first prefernce is given to local var and then global var.
- We can pass arguements to a function in 3 ways -

- Positional(default)/Value - def func(i, j =100)
  func(10,20)
  here, order of arguement matters and j=100 is its default value. if we don't provide any value of j during function call , it will automatically take values as 100.
  -Keyword arguements -
  def func(name, age)
  func(name="Pavan", age=19)
  Here, order of arguements doesn't matter
  -Mixed Positional and Keyword arguements
  we can mix positional and keyword arguements but make sure all positional arguements are given first and then keyword args. Once , we give one keyword args, then we can't give any positional args.
  func("Pavan", age = 19)
- Multiple values are returned as tuples.

File Handling in Python -

- We can R/W, Open/close, append , iterate thru file.
- we can open a file in 3 modes - r, w,a
  file = open("myfile.txt", "w") -- opened file in write mode
  file.write("haha\n")
  file.close
- Always close file at end of operation.
- Here file behave as pointer to the file.
- Reading/writing occurs in terms of lines.
- If we open an already existing file in write mode, it will be cleared first. If we want to avoid this, open in append mode.
- To read we have 3 methods -
  file.read([3]) - reads only 3 characters
  Note If no number is provided it will read entire file
  file.readline() - reads only first line
  file.readlines() - reads entire file as tuples(containing strings) of each line.
- To append , open in append mode and then file.write()
- To iterate thru file-
  file = open("myfile.txt", "r")
  for f in file:
  print(f)
  file.close()

OOP in Python

Classes and Objects -

- Class is an logical entity which contains variables and Methods, which contains the logic that can be performed over that class object.Class doesn't occupy space in memory.
- Object is an Physical Entity of any class. It occupies space in memory
- To create a class -
  a,b = 100, 200 # Global Variables
  class MyClass(): # Parenthesis are optional
  a, b = 10, 20 # Class Variables
  def func(self):# instance Method
  pass
  @staticmethod
  def add(a, b): # static method with parameter
  print(a+b) #This a&b are local variables
  print(self.a+self.b) # These are class variables
  print(globals()['a']+globals()['b'] # these are global variables

```
m1=MyClass() # Named Object
m1.func()
m1.add(3,5)

m2=MyClass() # Named Object
m2.add(7,8)

print(id(m1))
print(id(m2))

MyClass.add() # Nameless object + accessing static method
```

- Instance method can only be called after creating an object of that class but static method can be called without creating an object.
- we can create local, class, global variables with same name but to access them we need to differently.
- A function is created outside of a class, but a method is created inside a class and it belongs to that class
- Only instance method has default parameter as self, for static method we dont need to have self as parameter.
- To access class variables, use self.var
  To access global var, use globals()['var'] if we have both local and global var of same name. Else we can access directly.
  Local variables can be accessed directly as var
- c1 = MyClass()
  c2 = MyClass()

c3 = c1
here, c1 and c3 will point to same location in memory.
Also, c1, c2, c3 are called reference variables.

- we can compare 2 objects -
  print(c1 is c3) --> True
  print(c1 is not c3) -->False
- Constructor -
  It is a method that is used to initialize a class object. It is automatically called when we create an object.
  class MyClass:
  def **int**(self, var1, var2): # constructor with args
  self.var1 = var1 # converting local var to class var
  self.var2 = var2
  print(var2)
  def m1(self):
  print("this is m1")
  self.m2() # calling class method into other method
  def m2(self):
  print("this is m2")

c = MyClass(var1, var2)

- Local var are converted to class var so that other method can also use it.
- def **str**(self):
  this method executes automatically when we print reference variable
  Note - Return Type of this method should only be string
  If we don't overwrite this method, by default it returns a string containing address of object.
  -def **del**(self):
  this automatically gets execute when we detroy an object.
- An complete class Example -
  `

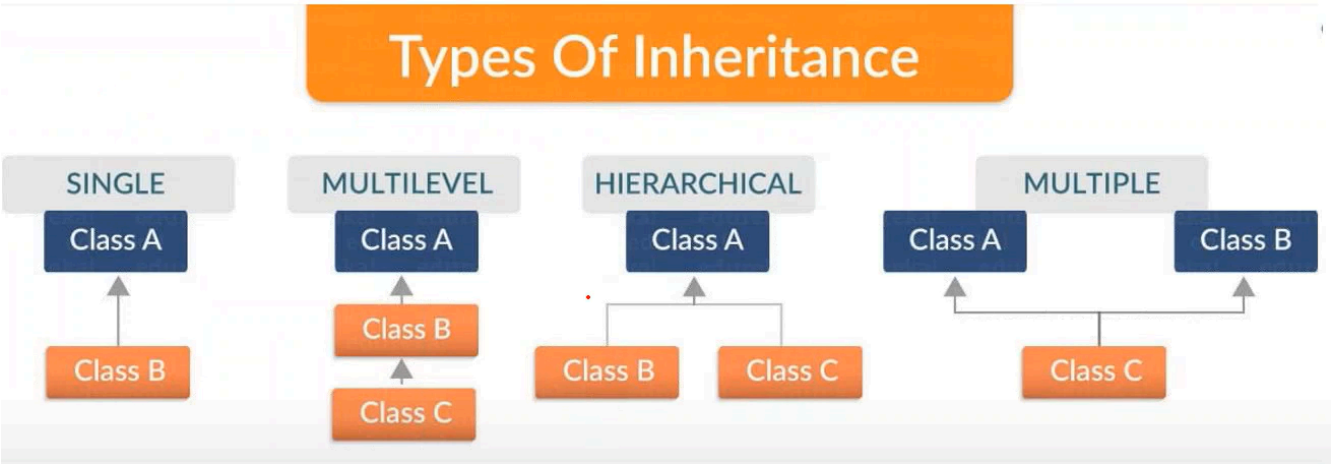
  class Emp:
  def **str**(self, Eid, Ename, Esal):
  self.Eid = Eid
  self.Ename = Ename
  self.Esal = Esal

```
        def__str__(self):
                        return ("Eid:() Ename:() Esal:()".format(Eid,
Ename, Esal))

        def __del__(self):
                        print("Destroyed")
```

```
e1 = Emp(101,"Pavan", 10000.00)
print(e1)
del e1
```

Inheritance in Python

- A class can access variables and methods of other class , if it inherits from it.
- Parent class can also be called as Base Class or SuperClass
- Child Class can also be called as Derived Class or SubClass.
- To inherit from a class A , we can define like -
  class MyClass(A):
- Types Of Inheritances -



- An Example of Hybrid Inheritance -
  class A:
  x,y= 1,2
  def m1(self):
  print(self.x+self.y)

```
class B(A):
a,b= 3,4
def m2(self):
print(self.a+self.b)

class C(A):
p,q= 5,6
def m3(self):
print(self.p+self.q)

class D(B, C): # D inherits from Both B and C
s,t= 7,8
def m4(self):
print(self.s+self.t)
```

a= A()
a.m1()

b= B()
b.m1()
b.m2()

c= C()
c.m1()
c.m3()

d = D()
d.m1()
d.m2()
d.m3()
d.m4()

- use of super() keyword in child class -
  super() can be used to invoke parent class -
  constructor, variables, methods. Eg. -
  a,b = 1,2
  class A:
  a,b = 10, 20
  def **int**(self):
  print("Constructor from A")
  def m1(self):
  print("this is m1 from A")

class B(A):
a,b = 5,10
def **init**(self):
print("constructor from B")
super().**init**(self) # invoked parent class constructor
A.**init**(self) # Another way to invoke parent class constructor

```
        def m2(self, a, b):
                print("this is m2 from B")
                super().m1() # invoked parent class method
                print(a+b) # local var
                print(self.a+self.b)  # class var
                print(super().a+super().b) # parent class var
                print(globals()['a']+globals()['b']) # global var
```

bobj = B()
bobj.m2()

- Also, if no constructor is present in child class then parent class constructor is called by default.

Polymorphism in Python

- It means a single method or variable of a class, can act in different ways based on the parameter that we pass while calling it.
- It is achieved by using Overriding or Overloading.
- Overiding means redefining a method or variable of parent class in child class(with the same name) but changing its logic or value respectively.
  Eg. -
  class Bank:
  name= "Bank"
  def ROI(self):
  return 0

class HDFC(Bank):
name="HDFC"
def ROI(self):
return 10.5
Here, if we don't override the name, then if we still access the name , it will give the name HDFC from parent class.

- Overloading means a method will act differently based on parameter we pass to it when calling.
  Eg -
  class Bird():
  def fly(self,name=None):
  if name== "Parrot":
  print("can fly")
  if name== "Penguine":
  print("cannot fly")
  if name is None:
  print("No Input")

obj=Bird()
obj.fly() # No input
obj.fly("Parrot") # can fly
obj.fly("Penguine") # cannot fly

Encapsulation in Python

- It means restricting access of a class variables or Methods outside the class by making it private.
- We can make variables or methods private by prefixing them with __
  Eg -
  class Empid:
  __empid = 101

```
    def __getempid(self, eid):
    self.__empid = eid
    def dispempid(self):
    print(self.__empid)
    def changempid(self, eid):
    self.__getempid(eid)

e1=Empid()
print(e1.__empid) # 101
print(Empid.__empid) # error b/c private variable
e1.__getempid # error b/c private method
e1.dispempid() # will display empid
e1.changeempid(105) # will change emp id
```

Abstraction in Python

- Abstract class are those class which have one or more abstract methods.
- Abstract Methods are methods, whose only declaration is given. Their logic/Implementation is not present in the class.
- We can't directly create objects of abstract classes.
  First we need to create a child class, where we implement the logic of all abstract methods of parent class and then we can create object of the child class.
- In python, all abstract classes inherit from a default class named ABC. It is present in abc module in python.
- If an abstract class has , say 2, abstract methods and in a child class if i only implement 1 abstact method, then this child class will still be considered a abstract class only. We need to have a child class that declares all the abstract methods , then only we can create objects of it.
- If the child class doesn't have a constructor, then constructor of parent class will be invoked while creating object for its child class
  Eg-
  from abc import ABC, abstraction method
  class A(ABC):
  def **init**(self, value):
  self.value = value
  @abstractionmethod
  def disp1(self):
  None

```
    @abstractionmethod
    def disp2(self):
                pass

class B(A):
def disp1(self):
```

```
print("this is disp1")
def disp2(self):
print("this is disp2")
def add(self):
print(self.value+100)
def sub(self):
print(self.value-10)

obj = B(100)
obj.disp1()
obj.disp2()
obj.add()
obj.sub()
```

Modules in Python

- A module is just an another python file.
- we can use its classes and methods in our main function by importing them
- There are 2 ways to import modules -
  1. import operations
     Here, to use operations file functions , use operations.add(10,20)
  2. from operations import add, mul
     now, use add(10,20) mul(10,20) directly
     If we want to import all functions of operations file -
     from operations import *
     and then use add(10,20) mul(10,20) directly
- If we import multiple modules ,using 2nd way, containing same function (say display), then the last imported module function will be given priority and used. To avoid this, import by way 1 and then use animal.disp() , bird.disp()
- If we have class in our module, to access its method we need to first create an object of that class and then use.
  Eg - Approach 1

```
class Animal: # in a.py
def disp():
print("I am animal")

        class Bird: # in b.py
                        def disp():
                                        print("I am bird")

import a
import b
```

```
obj1 = a.Animal
obj1.disp()

obj2 = b.Bird()
obj2.disp()
```

Approach 2 -
from a import Animal
from b import Bird

obj1 = Animal()
obj1.disp()

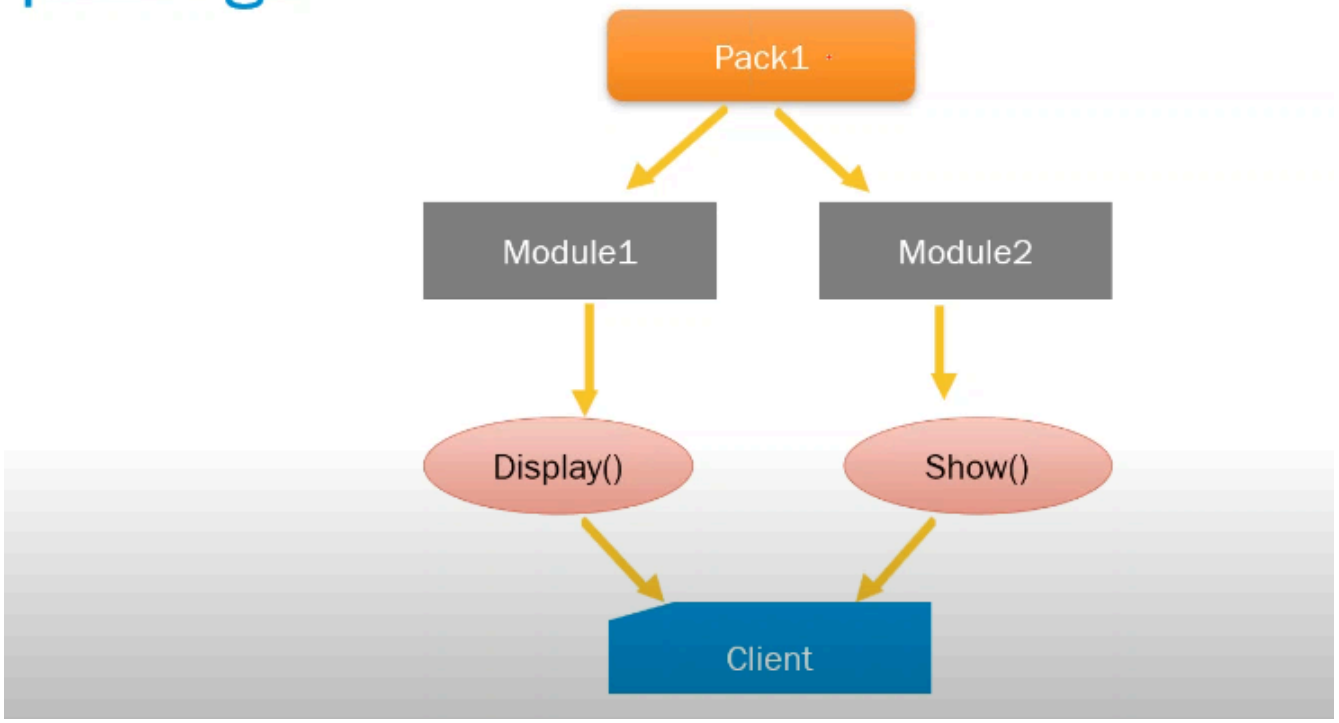
obj2 = Bird()
obj2.disp()

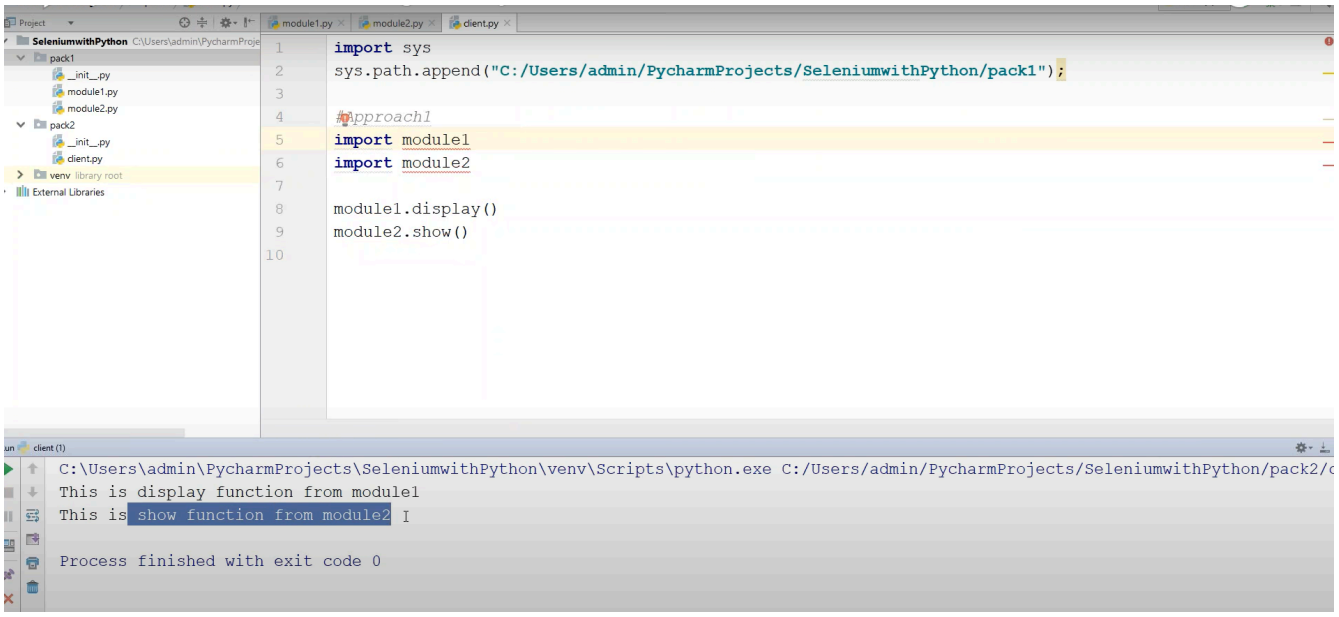
- If we want to know how many classes or functions are there in our imported module , we can use -
  import m
  print(dir(m)) -- this will give user created classes or functions, plus few default methods or functions of a python class.

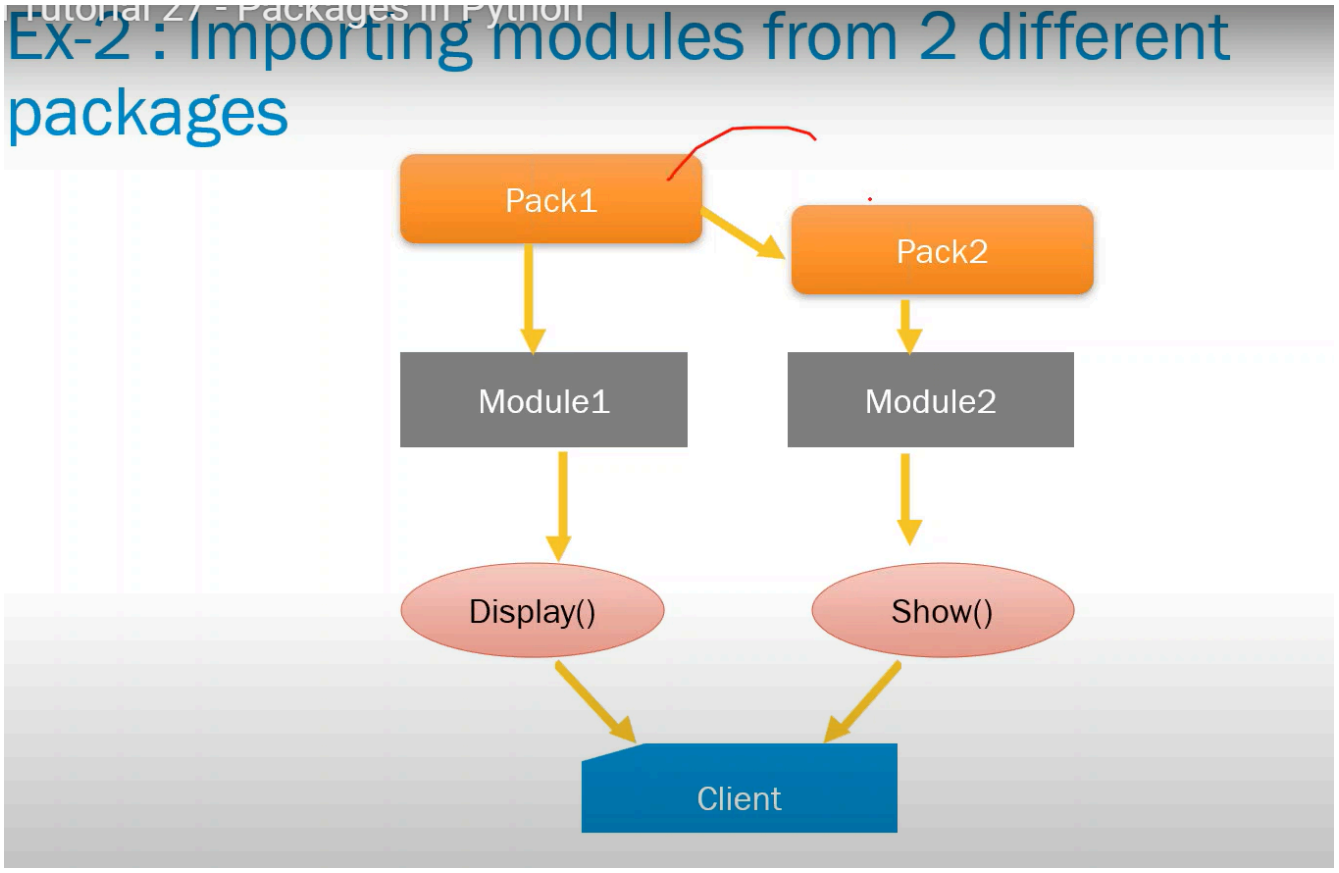
A Python Package

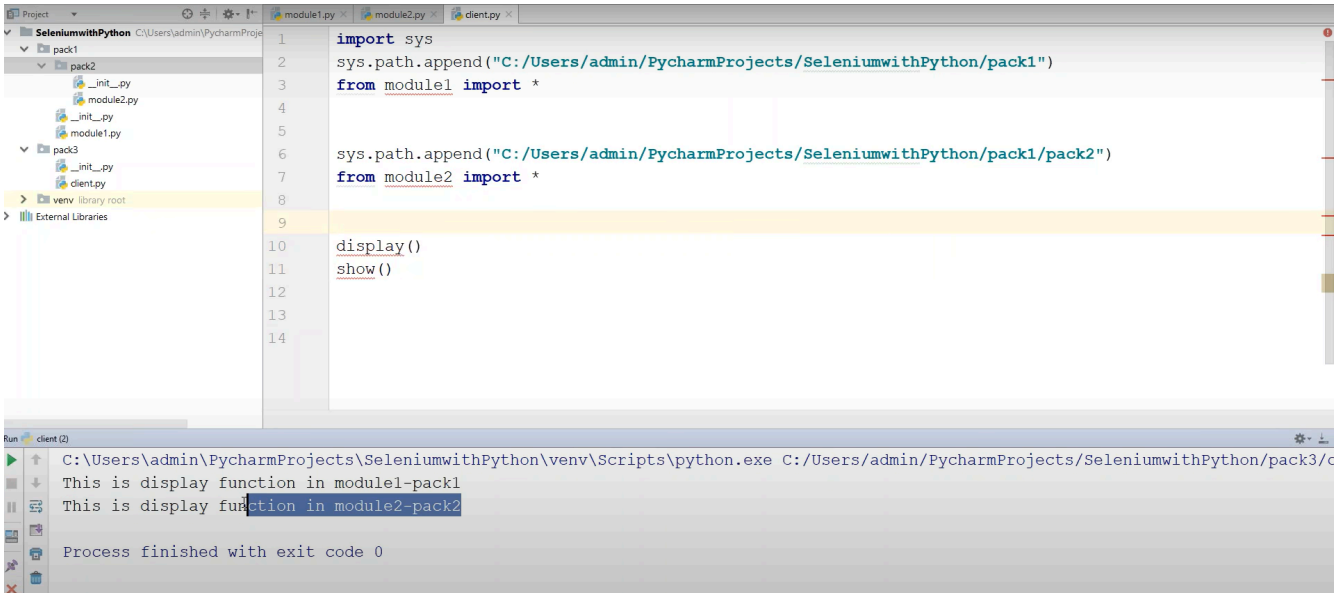
- A python Package is a folder that contains many modules and other folder, i.e. sub-packages
- We can import class and functions of other pakage's modules using -
  import sys
  sys.path.append($PATH_TO_PACKAGE)
  import module 1
  Then , we can use its class and functions
  Eg. -

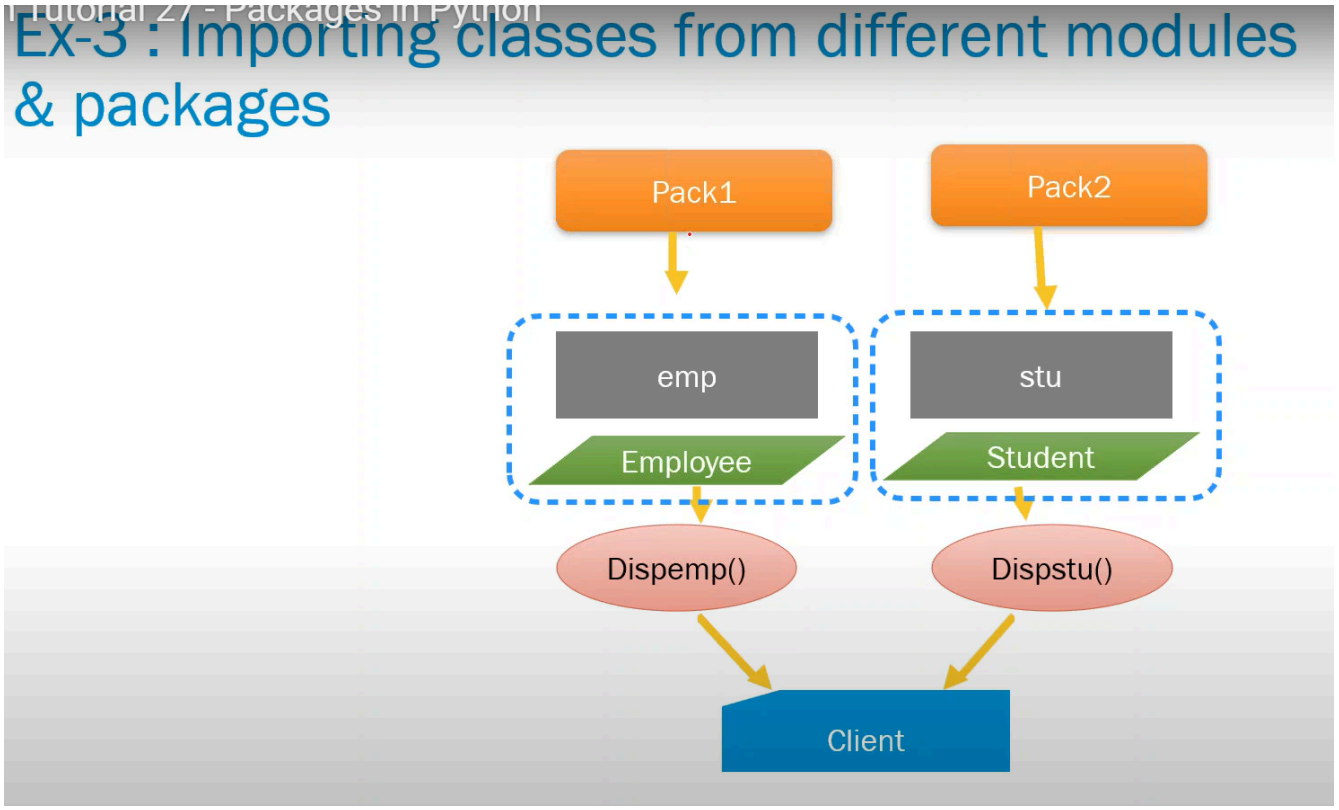
Ex-1 : Importing modules from Single package

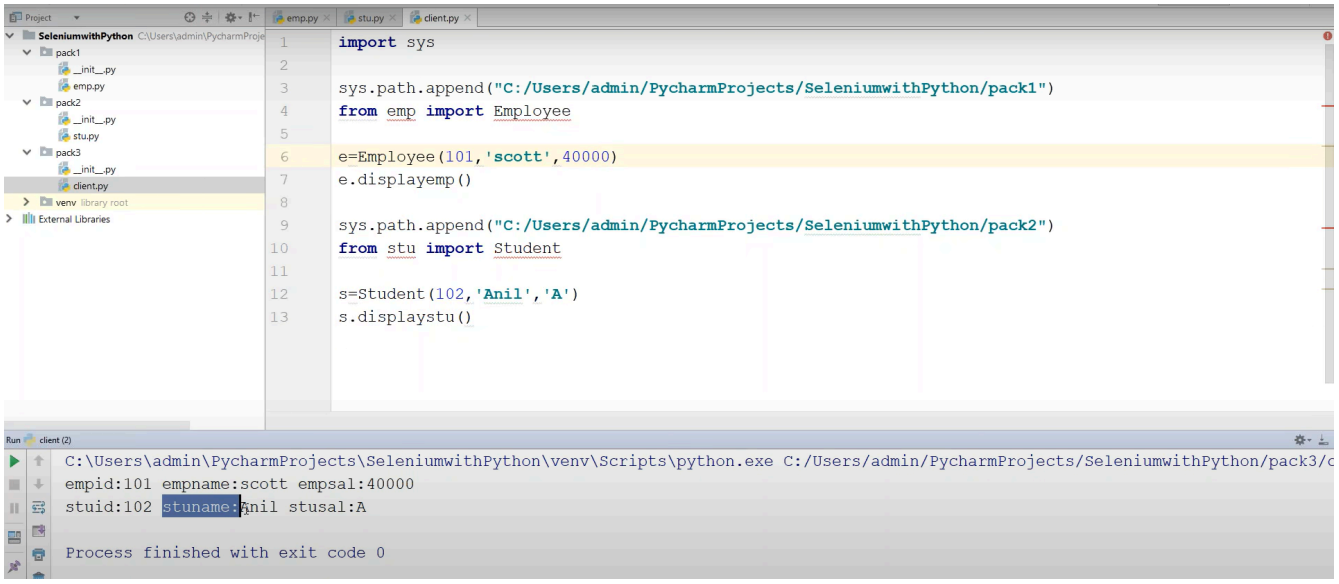
1. 



2. 
Tutorial 27 - Packages in Python
Ex-2 : Importing modules from 2 different packages
Pack1
Pack2
Module1
Module2
Display()
Show()
Client


import sys
sys.path.append("C:/Users/admin/PycharmProjects/SeleniumwithPython/pack1")
from module1 import *
sys.path.append("C:/Users/admin/PycharmProjects/SeleniumwithPython/pack1/pack2")
from module2 import *
display()
show()
C:\Users\admin\PycharmProjects\SeleniumwithPython\venv\Scripts\python.exe C:/Users/admin/PycharmProjects/SeleniumwithPython/pack3/c
This is display function in module1-pack1
This is display function in module2-pack2
Process finished with exit code 0

3. 



Exception Handling in Python

- An Error is either a programming syntax error or logical error in the program, whereas Exception is an abnormal condition that occurs due to illegal user input.
- If an exception is raised at any point in program, rest of program will not be executed.
- To handle , this we can use try, except, else, finally block.
- try block contains statement that can throw error
  and there can be multiple except block, which will
  handle exception based on the type of exception error
  Syntax - except :
  else block is executed if try ststement doesnt throw any
  error.
  finally block will always be executed no matter what so if exception occured and there are no except

staments , so finally will be evaluated.
Eg. -
try:
print(10/0)
except ZeroDivisionError:
print("Executed Zero Division Error")
except NameError:
print("Executed Name Error")
else :
print("Try block didn't throw any error, else block executed")
finally:
print("Finally block....Always Execyted")

- we can also use exception thrown as an object and then print the type of error thrown
  Eg. -
  try:
  print(10/0)
  except as ex:
  print("Exception :", ex)
- We can also rasie our own Exceptions using -
  raise ("Message to Print")

Lambda Function

- An variable can also act like a function,using lambda keyword, if it has only one line of expresssion.
  Eg - syntax - lambda arguements: expression
  x = lambda a : a+10 #x is a function, a is passed as parameter
  print(x(5))
- We can give any number of arguements, but there should a single expression.

*args and **kwargs in Python

- *args is used to pass arbitrary number of inputs to a function.
- **kwargs is used to pass arbitray number of key-value pair to a function.
- we can use * a and **a also, but convention is like *args and **kwargs.
  Eg . -
  def sum(*args):
  sum = 0
  for i in args:
  sum+=i
  print(sum)

sum(10,20, 30, 40)

- We can also pass list as arguments to function as -
  def disp_three(a,b,c):
  print(a,b,c)

```
args = [10,20,30]
disp_three(*args)
```

- Eg -
  def disp_three(a,b,c):
  print(a,b,c)

```
a = {name="tom", sport="football, roll="10"}
disp three(**a)
```

Eg -

```
def disp(**a):
for i, j in a.items():
print(i,j)
```

```
a = {name="tom", sport="football, roll="10"}
disp three(**a)
```
