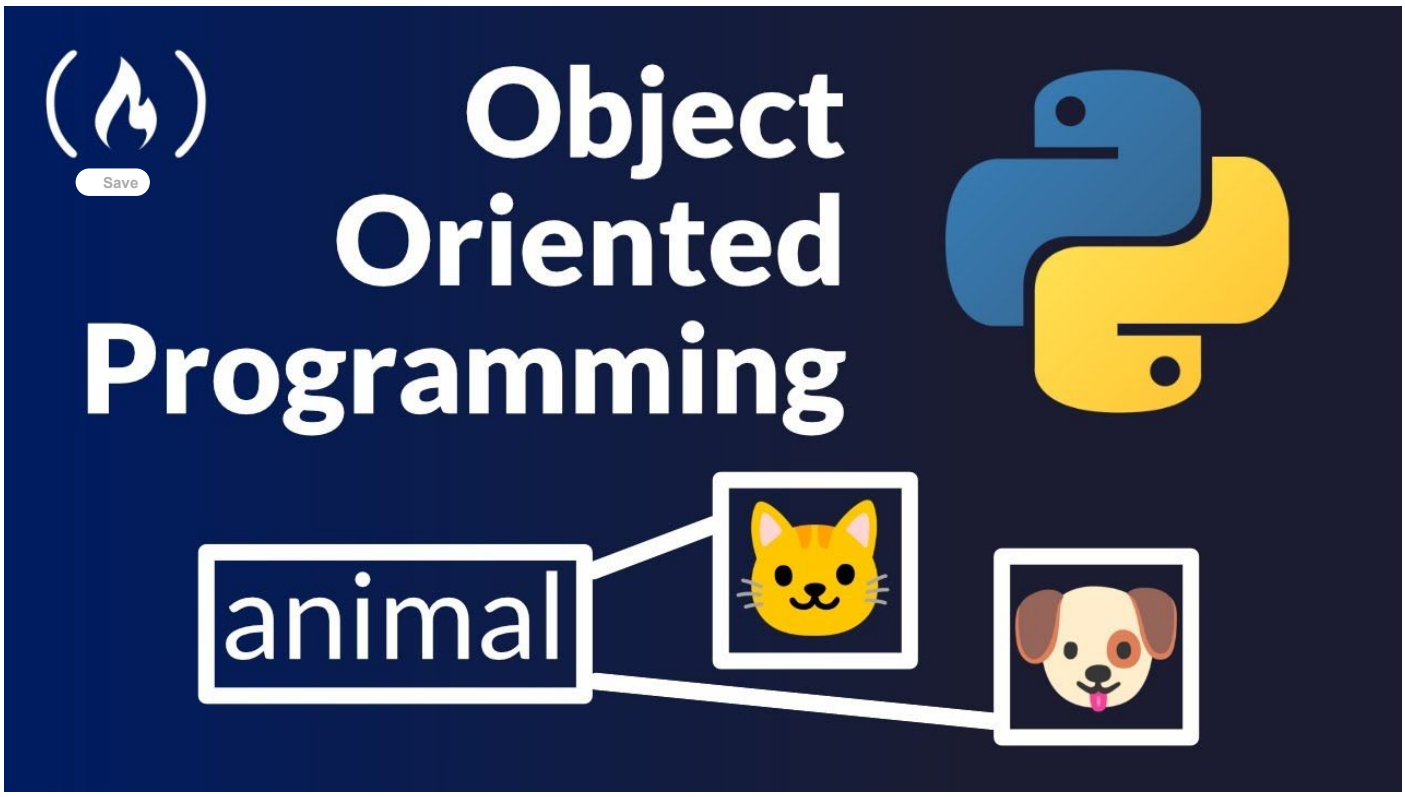
Object-Oriented Programming



Hello everyone, welcome to the Python OOPS assignment.

Table of Contents:

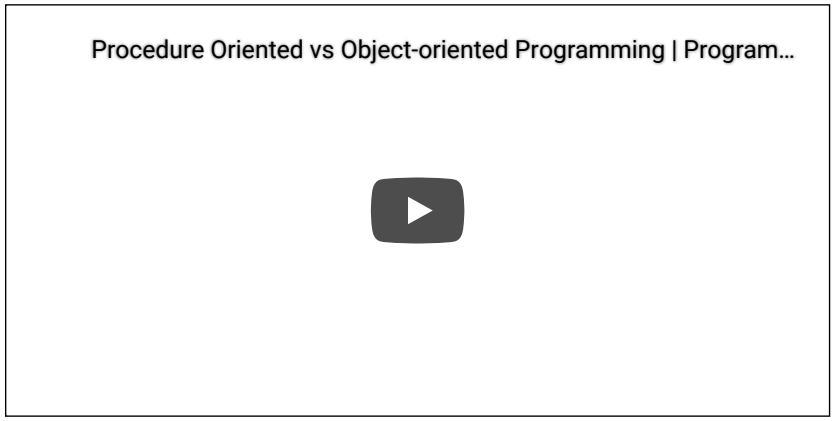
- Differences between Procedural and Object-Oriented Programming??
- Why we need Object-Oriented Programming??
- What is Object-Oriented Programming?
- What are classes?
- What is an object?
- What are member variables and methods?
- What are Attribute?
- What is instantiating a class?

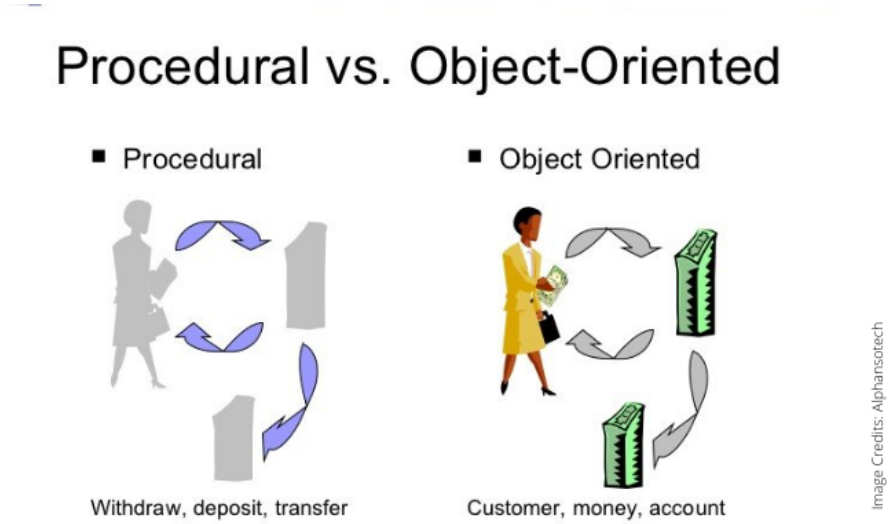
And many more Question will be no more in your head!!!!

Come dive with me...

What is the difference between procedural programming and object-oriented programming?

```
###Refer Video
from IPython.display import YouTubeVideo
YouTubeVideo('-y8OTRJ7Cvo', width=600, height=300)
```





Procedural Programming

- `basic unit are functions`
- programming in these type of procedural languages involves choosing a data structure and
  - then designing algorithm and
  - then translating that algorithm into a code

`Shortcomming of only using fucntions are:`

- Data saved as Global Data need to be send to fucntions like openApp() and other fucntions
- These fucntion takes data as Arguments or Global Variables
- These fucntion are Passive i.e. cant hold any data inside them

```
"""
pseudo code for Procedural Programming
# travel

Global Data
```

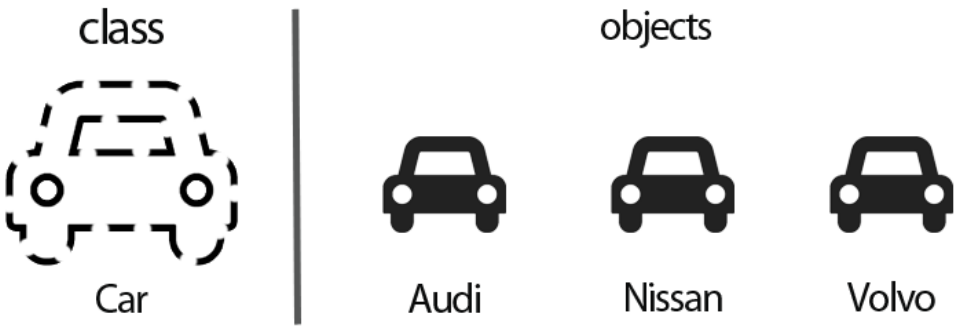
```
travel {
    openApp()
    bookCab()
    waithForTheCab()
    sitInTheCab()
    reachDestination()
    PayCabFare()
}
"""
```

```
'\npseudo code for Procedural Programming\n# travel\n\nGlobal Data\n\ntravel {\n    openApp()\n    bookCab()\n    waithForTheCab()\n    sitInTheCab()\n    reachDestination()\n    PayCabFare()\n}\n'
```

Diff. in Method and Function

Object Oriented Programming

- Basic unit is `Class/Object`



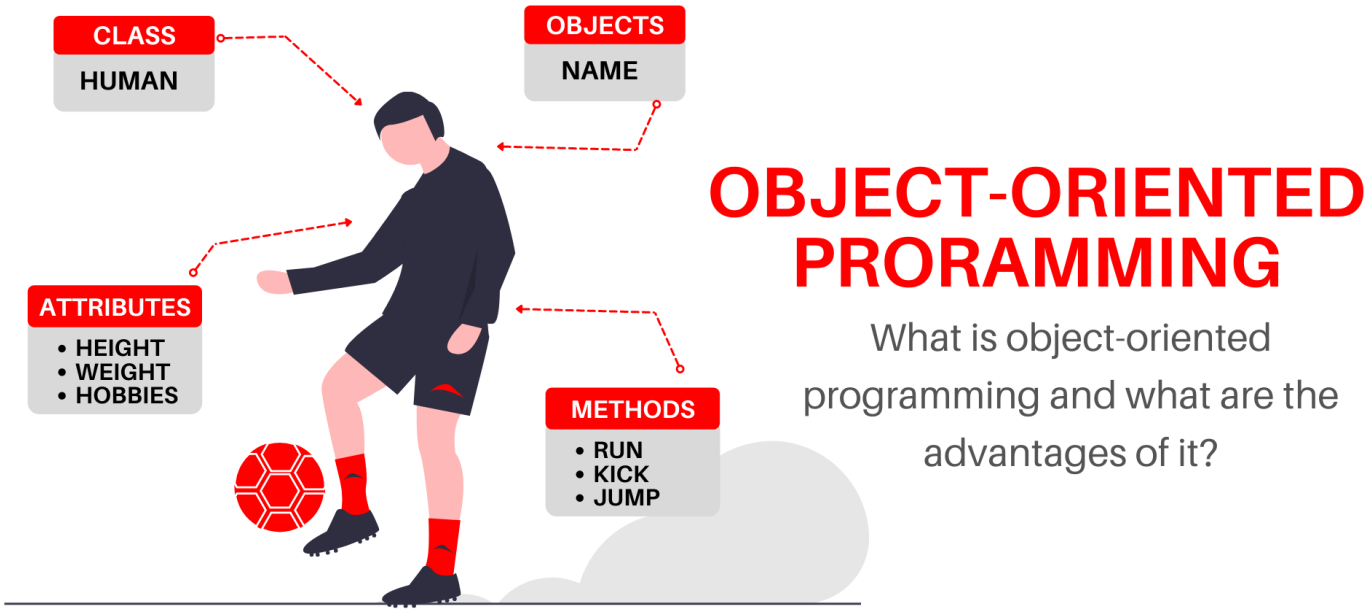
`Class` => blueprint of `Data/Attributes/Member Variables` & `Methods`
`Methods` => function inside a class are called methods
`Data Member|Attributes|Member Variables` => variables containing data inside a class **`Declaring Objects (Also called instantiating a class)`**
`Object` => single unit that `combines data` (Class Variables, Methods, Attributes) `and methods` (function inside class)
`Attributes` => Data inside an object

Class and Object Difference

- A `variable` stored in an instance or class is called an attribute
- A `function` stored in an instance or class is called a method

According to Python's glossary:

- `Attribute` : A value associated with an object which is referenced by name using dotted expressions. For example, if an object o has an attribute a it would be referenced as o.a
- `Method` : A function which is defined inside a class body. If called as an attribute of an instance of that class, the method will get the instance object as its first argument (which is usually called self). See function and nested scope.



Method definition

Refer: https://www.geeksforgeeks.org/python-classes-and-objects/

```
###Refer Video
YouTubeVideo('qiSCMNBIP2g', width=600, height=300)
```


```
# demonstrate instantiating a class

#instantiate a class Dog

    # Class Attributes
    #set attribute 1 to "mammal"
                  # <= ***** Data ***** i.e. Attributes / Member Variables
    #set attribute 2 to "dog"


    # Class Method (method definition have self in it)
                     # <= ***** Methods ***** i.e. Function inside class
        #print first attribute

        #print second attribute
```

```
# Driver code
# OBJECT Instantiation


# Accessing Class Attributes and Method through OBJECTS

#apply defined function to the created object
```

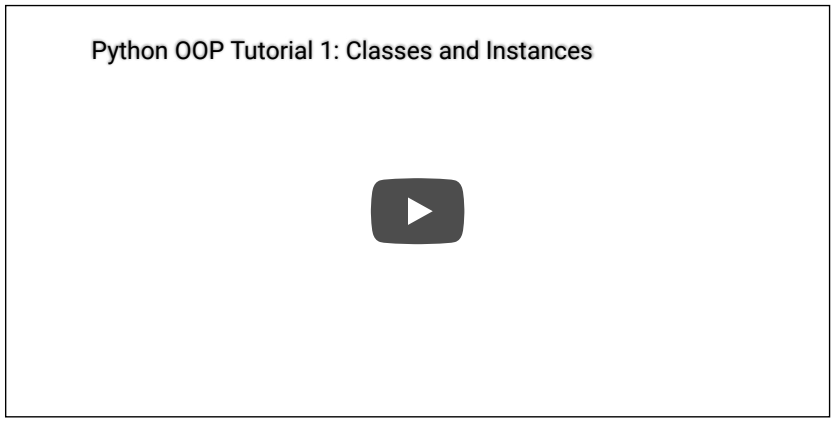
```
 mammal
  ********************
 I'm a mammal
 I'm a dog
```

```
"""
pseudo code for Object-oriented programming

class cab{
    cabService, cabMaker, location, cab_number  <= ***** Data ***** i.e. Attributes
    book(), arrival(), start()                  <= ***** Methods ***** i.e. Function inside class
}

class cabDriver{
    name, employeeId     <= Data
    openDoor(), drive()  <= Methods
}

class passenger{
    name, address     <= Data
    openApp(), bookCab(), walk()  <= Methods
}

"""
```

```
 '\npseudo code for Object-oriented programming\n\nclass cab{\n    cabService, cabMaker, location, cab_number  <= ***** Data *****
 i.e. Attributes\n    book(), arrival(), start()                  <= ***** Methods ***** i.e. Function inside class\n}\n\nclass cab
 Driver{\n    name, employeeId     <= Data\n    openDoor(), drive()  <= Methods\n}\n\nclass passenger{\n    name, address     <= Da
 ta\n    openApp(), bookCab(), walk()  <= Methods\n}\n\n'
```

Creating and Runing Python Class-1

```
YouTubeVideo('ZDa-Z5JzLYM', width=600, height=300)
```



`Class` => keyword

```
# define an empty class

    #pass


# ford,honda,audi is an INSTANCE of Class car or
```

```
## OBJECT of class car
              # <= Object/instance-1
              # <= Object/instance-2
              # <= Object/instance-3


# associating data to above objects [adding attributes to above Objects]
#add values to speed attribute to above created objects
#Example: ford.speed = 200
```

Attribute => speed is Attribute above

If we have empty Class like above then we can add Attributes on the fly

See we are adding speed attributes after class declation and object creation

```
# associating more data to above objects [adding more attributes to above Objects]
```

```
# printing attributes
```

```
100
red
```

```
# changing ATTRIBUTES of audi OBJECT
```

```
# printing attributes
```

```
400
red
```

These speed and color are variables which contains some data in them, but still we have not added any behaviour's or method's to our Class car

behaviour => i.e. Methods and How to use them to manupulate data which we have provided in form of attributes

Creating and Runing Python Class-2

```
# defining empty class
```

```
# making object or instance of class
```

```
# adding attribute(height,width) to above object/instance
```

```
# printing are of rectangles (hight * width)
```

```
4000
1000
```

Still we have not added any `behaviour` to our class

behaviour => i.e. `Methods` and `How to use them to manupulate data which we have provided in form of attributes`

Adding Behaviour(Methods) to Class(to use them to manupulate data)

I Will add Behaviour i.e. Methods to class and see how to manupulate data with it

Use of `__init__ method` and `self keyword`

`__init__ method` used as a `constructor for class`

- Usually `__init__` does some initialization work e.g. initialize attributes and other functions
- Arguments passed to class name are given to its `__init__() method`
- It would be incorrect to call `__init__ constructor of class`
- But **init** is closest thing we're going to get in Python to a constructor

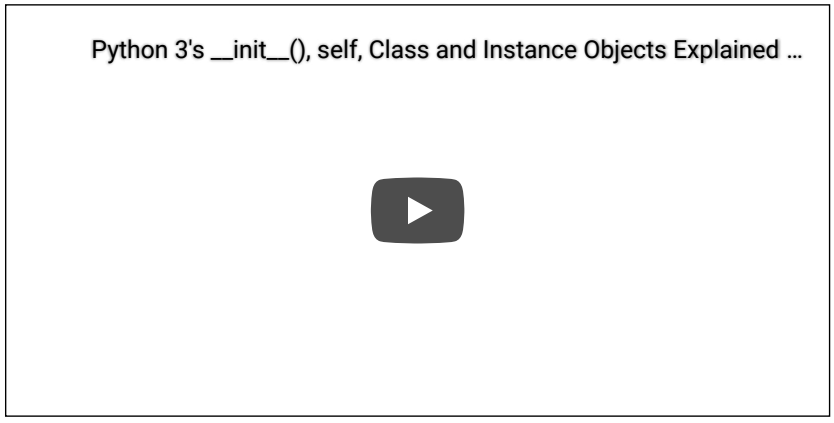
NOTE

init serves as an `Constructor`, it is not an constructor but it behaves as one

`self keyword`

- self is first argument of every method
  - IT is a reference to current instance of class
- `self keyword` is similar to `this` keyword in Java or C++

```
YouTubeVideo('AsafkCAJpJ0', width=600, height=300)
```




```
################### CONSTRUCTOR DEFINIATION###################
"""
Constructor is a method that is called right when we instantiate an object of a certain class.

================================================
Practical example:
(this example is going to be written in pseudocode, as python programming language does not have a proper constructor itself)

class Person:
          constructorMethod(name):
              print('Hello, ' + name)

me = Person('Mukesh Manral')
================================================

==> Code above is basically 'creating' a Person object called "me".
==> Constructor method is defined inside class Person, and because of that,
```

```
    ==> it is going to be called at the moment that we created object "me".

This is why this method is called "constructor",
    because it is basically used to initialize things right when we create an instance of class that we're
    currently working on i.e. it "constructs" attributes of object.

If we run above code, it is going to give output: 'Hello, Mukesh Manral', because inside constructor method it says
print('Hello, '+ name), and name that I passed as parameter was Mukesh Manral, which is my name.
"""
```

```
'\nConstructor is a method that is called right when we instantiate an object of a certain class.\n\n============================
==================\nPractical example: \n(this example is going to be written in pseudocode, as python programming language does
not have a proper constructor itself)\n\nclass Person:\n          constructorMethod(name):\n              print(\'Hello, \' + nam
e)\n\nme = Person(\'Mukesh Manral\')\n================================================\n\n==> Code above is basically \'creating\'
a Person object called "me". \n==> Constructor method is defined inside class Person, and because of that, \n    ==> it is going t
o be called at the moment that we created object "me".\n\nThis is why this method is called "constructor", \n    because it is bas
ically used to initialize things right when we create an instance of class that we\'re \n    currently working on i.e. it "constru
cts" attributes of object.\n\nIf we run above code, it is going to give output: \'Hello, Mu  '
```

```
# defining class

    """
    * defining an __init__ method with self keyword as an first default argument
    * __init__ method serves as an constructor for this class

    * usually used to initealise some ATTRIBUTES or some FUNCTIONS, because this is first method
    which is called during instantiation of a Class
    """



# ford,honda,audi is an INSTANCE of Class car or
## OBJECT of class car
              # <= Object/instance-1
              # <= Object/instance-2
              # <= Object/instance-3

# 3 instances created
```

```
__init__ is called
__init__ is called
__init__ is called
```

As we have created 3 instances of car class, now **init** method will be called 3 times, because whenevery we call some instance of class the first method called is **init** method

So everytime one instance is created, **init** methos is called

NOTE

init serves as an `Constructor`, it is not an constructor but it behaves as one

```
# defining class



        """
        provided 2 argument to __init__
        named as speed, color
        """



# making instance | Object of class
```

```
45
Black
__init__ is called
60
White
__init__ is called
50
```

```
Grey
__init__ is called
```

We did not provided arguments at the time of Initealization(instantiation) of class, as 2-Arguments wer required as stated inside **init** method

```
# defining a class
```

```
# providing arguments while instantiating
```

```
300
red
__init__ executed
400
blue
__init__ executed
500
black
__init__ executed
```

Usually these arguments are given when you initialize the value of speed or color.

```
# accessing ford instance|object speed and color
#print(ford.speed)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-19-d0e6d5f72435> in <module>()
      1 # acessing ford instance|object speed and color
----> 2 print(ford.speed)
      3 print(ford.color)

AttributeError: 'car' object has no attribute 'speed'

SEARCH STACK OVERFLOW
```

Error says we have provided Attributes named as speed and color(**init**(self,speed,color):) but not assigned speed and color to any Attribute(variable) inside class car i.e.like ford.speed = 200

Asigning speed and color using **self keyword**, it will assign value to current object

```
# defining a class
```

```
        #print(speed)

        #print(color)

        # here self is the current object by itself (Initalization of Attributes(speed,color) using __init__ method)
        #initialize with self of speed and color
```

```
# providing arguments while instantiating with speed and color [3 examples]
```

```
"""
# manual initealization which is removed now cause
# __init__ is being used for Attributes initialization
ford.speed = 300
ford.color = 'red'
"""
```

```
# acessing ford instance|object speed and color
```

```
300
red
__init__ executed
400
blue
__init__ executed
500
black
__init__ executed
300
red
```

Defining Multiple Constructors

`__init__` behaves as an constructor, but it is not a constructor

Lets see if it is possible to define multiple constructors in Python?

```
# not giving __init__, self as an argument
#define class car


        #print __init__ executed


# initealizing a class
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-0a7d66fcccfd> in <module>()
      7
      8 # initealizing a class
----> 9 ford = car()

TypeError: __init__() takes 0 positional arguments but 1 was given

SEARCH STACK OVERFLOW
```

What Error is `__init__() takes 0 positional arguments but 1 was given`

- self is passed in automode when you initealize a class, that is why it is saying 0 positional arguments but 1 was given

```
# now giving __init__ self as an argument
#define class car

    #define init function with self argument



# initealizing a class
```

```
__init__ executed
```

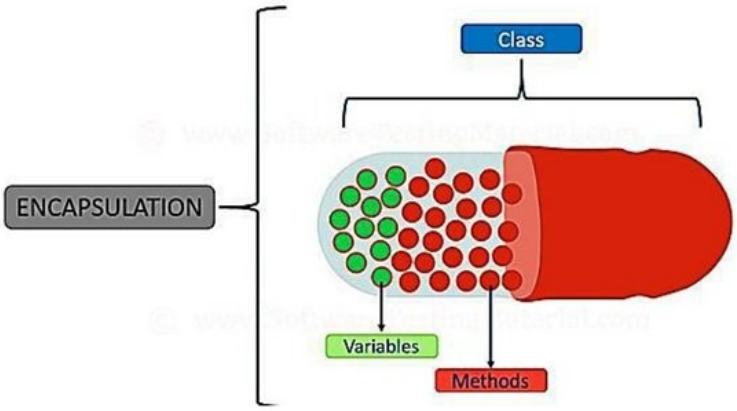
Is it possible to define multiple `__init__` method inside a class i.e. Multiple Constructor inside a class

```
# defining two  __init__ methods inside car class

    #def 1st init self and name


    #def 2nd init self and name
```

```
                    #considered as main init -- overwriting previous __init__

# instantiation

     Second __init__

# defining two  __init__ methods inside car class
    #def 2nd init self and name

    #def 1st init self and name

                    #considered as main init -- overwriting previous __init__

# instantiation

     first __init__
```

See above If you used mutiple `__init__` methos inside a class, which is not a valid implimentation, then last defined `__init__` method will be considered as main init and rest will be overwritten.

Encapsulation



Encapsulation allows programmer to:

- group data and subroutines that operation on them together in one place, and
- to hide irrelevant details from user

Making objects and algorithms invisible to portions of system that do not need them.

- Encapsulation allows us to isolate major design decisions, especially ones subject to change
- Encapsulation promotes code reuse

Will see `How to make your member variables private?`

- A double underscore makes variable Private which makes it harder to access but still possible
- Variable with one leading underscore is semi-private

Using Encapsulation in Python

```
YouTubeVideo('GN1LR0UoFI4', width=600, height=300)
```


```
#define class car

    #def init with self,speed,color

        #self with speed[self.speed=speed]

        #same for color


#passing the parameters speed and color



# updating ford speed



     1000
     red
```

```
#define class car

    #def init with self,speed,color

        #self with speed[self.speed=speed]

        #same for color


#passing the parameters speed and color


# updating ford speed with string which will lead to code break maybee



     It will lead to code break
     red
```

If anyone came and gave some string values to ford.speed which will lead to code break and maybe some loss, so we require `Encapsulation`

- it is very important to protect our data and only give acess to required data to other user
- it is very important when you want ot give your code to other people and they want to change it

To Encapsulate code we use Functions

```
#define class car

    #def init with self,speed,color

        #self with speed[self.speed=speed]
```

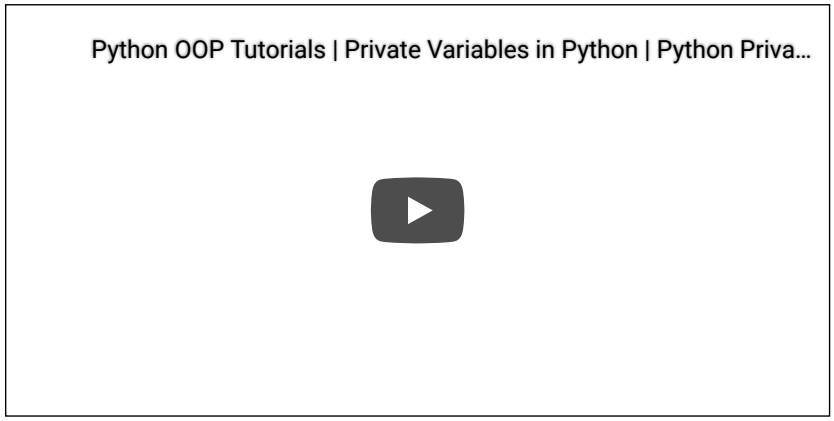
```
        #same for color

    '''
    Using function to
    Encapsulate out data
    '''
    # define a function to set value of speed


    # define a function to get value of speed

                           # Returning set speed

    '''
    Above is the use of getter and setter function for speed Attribute
    '''

#passing the parameter of speed and color



'''
updating ford speed using getter and setter fucntions which are as set_speed and get_speed [ford => object]
'''




 300
 red
```

Private Attributes

We need to protect our Attribute assignments which takes place in the fly i.e.(ford.speed = 300), shown in bellow code

- We have make out Attributes Private

```
YouTubeVideo('ogWtR_DIGKs', width=600, height=300)
```




```
#define class car

    #def init with self,speed,color

      #self with speed[self.speed=speed]

        #same for color

    '''
    Using function to
    Encapsulate out data
    '''
    # define a function to set value of speed
```

```
    # define a function to get value of speed

                          # Returning set speed

    '''
    Above is the use of getter and setter function for speed Attribute
    '''

#passing the parameter of speed and color


# updating ford speed using getter and setter fucntions which are as set_speed and get_speed [ford => object]


# updating attribute(speed) in the fly by using assignment operator (=)
# to protect this we will make it private attribute
```

```
400
red
```

Making speed a Private Attributes

Other OOPs languages have keywords like:

- public
- private
- protected

  In order to make there attributes say above

Python dont have these keywords so we use this convention => `__AttributeToPrivate`

```
# making data private hellow code


        """
        Inside this hellow class making three
        MEMBER Variables
        """

                        # _ <= Partially Private Attribute
                        # __ <= used to make a Attribute|Variable PRIVATE


# use of hellow method to acess above values
```

```
10
100
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-29-38bb6debdd6a> in <module>()
     15 print(Hellow.a)
     16 print(Hellow._b)
---> 17 print(Hellow.__c)

AttributeError: 'hellow' object has no attribute '__c'
```

SEARCH STACK OVERFLOW

`__c` when used ouside of class it through error saying 'hellow' object has no attribute '__c'

- Point is `__ <= is used to make a Attribute|Variable PRIVATE`
- This is a `convention` as there are no keyword like other oop's languages like C++ or Java

_ <= i.e. one undescore is for `Partially Private Attribute`

Now going to car Attributes and making them PRIVATE

```
#define class car

    #def init with self,speed,color

                               # __ <= used to make a Attribute|Variable PRIVATE
                               # __ <= used to make a Attribute|Variable PRIVATE
    '''
    Bellow is the use of getter and setter function for speed Attribute

    Using function to Encapsulate out data
    '''
    # define a function to set value of speed



    # define a function to get value of speed

                          # Returning set speed

        # define a function to set value of color



    # define a function to get value of color

                          # Returning set color

    '''
    Above is the use of getter and setter function for speed Attribute
    '''

#create 3 car objects by passing speed and color values



# updating ford speed using getter and setter fucntions which are as set_speed and get_speed [ford => object]


# trying to updating attribute(speed) in the fly by using assignment operator (=) which is not possible now
# after protecting it using __ and made it PRIVATE Attribute


#print speed and color of any one object
```

```
300
orange
```

- See above that now speed is not changed with `ford.__speed = 400` i.e on the fly also, we need to use `set_speed` function to do that

We have lenered to `set Private Member Variables` using `__` i.e. double underscore

`Private Member Variables` done, now learn to `Declare Private Methods`

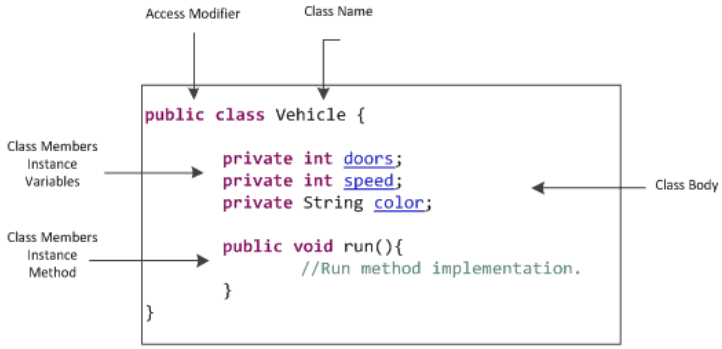
Declare Private Methods

To create a `Private Method` in a class you can use double underscores in front of your member, then they are made local to the class.

What is Private Member Variables?

It is private to the class i.e. you can use that Private Member Variables inside class but

- the movement you try to use or acess it `outside the Class it will give error`
- can say Private Member Variables are in-visible to our Object|Instance



How to use Private Member Variables inside class?

```
#define class hello

    #define init function

        #set a,b,c values



    # trying to acess private memeber variable inside class by making a new method
    #make new method

        #declare public variable

        #print("It's public")


#create object of above class,pass name


#apply decalred function to the object

#access the declared variables


# print(Hello.__c)
```

```
It's public
10
100
```

No error in above code tells that you can use Private Member Variables inside class or any Method of the class

Public Member Variable

`a` is a public memeber varible as it does not have any underscore in it

- It concludes that `a` which is a public memeber varible can be used inside or outside the class

Declaring Private Method inside Class

Simply use double underscore(__) in front of Method

Bellow is the code

```
# Declaring Private Method inside Class
#declare a class

    #define init function

        #declare 3 variables



    # trying to access private memeber variable inside class by making a new method

        #try to access public variable


    """
    Declaring a private method inside a class
    """
    #def __private_method(self):

        #print('private')


# instance | Object | instantiation of class




# using private method outside class (though will not work)
```

```
10
100
It's public
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-32-41b6e86f5ee4> in <module>()
     26 Hello.public_method()
     27 # using private method outside class (though will not work)
---> 28 Hello.__private_method()

AttributeError: 'hello' object has no attribute '__private_method'

SEARCH STACK OVERFLOW
```

You can acess private method inside class by using `self keyword`

Lets see how

```
# Declaring Private Method inside Class
#declare a class

    #define init function

        #declare 3 variables using self
```

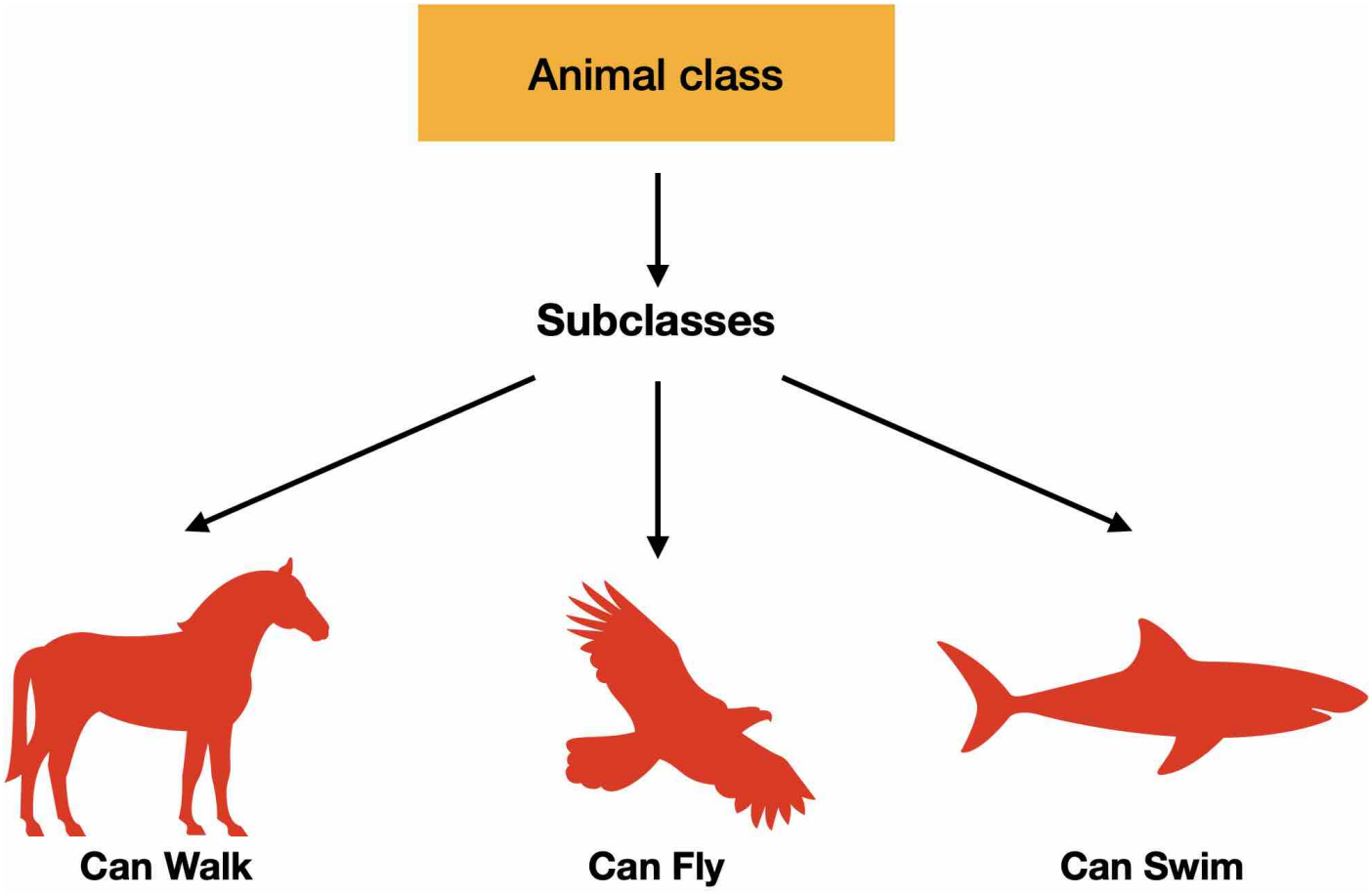
```
    # trying to acess private memeber variable inside class by making a new method



        """
        Acessing private method inside class step-2
        """



    """
    Declaring a private method inside a class step-1
    """



# instance | Object | instantiation of class

#print(Hello._b)

#Hello.public_method()
```

```
100
10
1000
It's public
private
```

Inheritance

Classes in Python can be extended

- creating new classes which retain characteristics of base class and this process is known as `Inheritance`



Python Inheritance Involves two kind of classes a `Superclass` and a `Subclass`

- Subclass inherits members of Superclass, on top of which it can add its own members.

Idea of inheritance is that a class can be defined to borrow behavior (member variable and member methods) from another class.

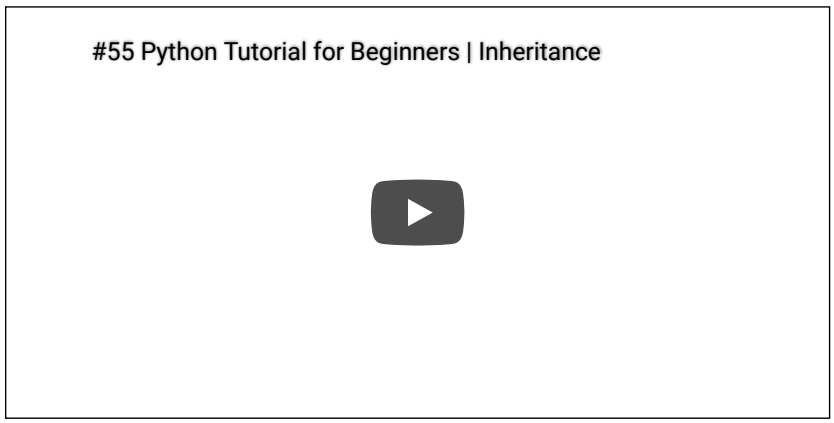
- When we define a subclass we place name of parent(superclass) in parentheses after subclass's(child) name on first line of definition

  '"class Parent(Child):"'

See Superclass and Subclass as Mobile(Superclass) and iphone,lava,samsung(Subclass)

- Relationship between Superclass and Subclass is shown by **is a** relationship
  - iphone isa Mobile

```
YouTubeVideo('Cn7AkDb4pIU', width=600, height=300)
```




```
#define class Mobile

    #create a private variable having value as None
                  #none keyword is used when we want to assign nothing to the variable
    #create another private variable having value as None
                  # __ <== to make variables private

    #define function set_values taking imei and model as parameters

        #assign imei value to the first private variable declared above

        #assign model value to the second private variable declared above


'''
Mobile class is superclass -- above
'''
'''
Iphone and Lawa are subclasses -- down
-----------------------------------------------------------------------
To inherit properties of Superclass into Subclasses,
* give Superclass name as an argument to Subclass like ==> Iphone(Mobile)
    i.e. Iphone class is Inheriting from Mobile class
'''
#define Class Iphone

    #define iphone_detail function

        #return self.__imei,self.__model


#define Class Lawa

    #define lawa_detail function

        #return self.__imei,self.__model


'''
```

```
NOTE: Now after Inheritance, Superclass Variables and methods can be accessed using Subclass
'''
# Iphone class instance

# Lawa class instance


# we can acess set_values methos using iphone instance
#iphone.set_values('1212sadasw145','Iphone-XR')


# now to see detailes of Iphone or Lawa we can call respective methods of their class
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-34-40e2a78e35c6> in <module>()
     39
     40 # now to see detailes of Iphone or Lawa we can call respective methos of there class
---> 41 print(iphone.iphone_detail())
     42 print(lawa.lawa_detail())

<ipython-input-34-40e2a78e35c6> in iphone_detail(self)
     20 class Iphone(Mobile):
     21     def iphone_detail(self):
---> 22         return self.__imei,self.__model
     23
     24 class Lawa(Mobile):

AttributeError: 'Iphone' object has no attribute '_Iphone__imei'

SEARCH STACK OVERFLOW
```

ERROR: as `__imei and __model` are Private member variables of Superclass i.e. Mobile class and cannot be accessed by Subclass

Point is ==> Private member of Superclass can't be inharited by Subclass

Solution is to make a getter method for private members

```
'''
Making a getter method to acess private member of Superclass from Subclass
'''
#define class Mobile

    #create a private variable having value as None
                #none keyword is used when we want to assign nothing to the variable
    ##create another private variable having value as None
                # __ <== to make variables private

    #define function set_values taking imei and model as parameters
                            # setting values
        # #assign imei value to the first private variable declared above

        #assign model value to the second private variable declared above


    #define function get_imei
                      # getting values
        #get imei value using self
        #return self.__imei

    #define function get_model

        #get model value using self

'''
using getter method in place of private member variables like __imei, as to inherit Superclass using Subclass
'''
#define class Iphone

    #define function detail

        #return self.get_imei(),self.get_model()
```

```
#define function Lawa

    #define function detail

        #return self.get_imei(),self.get_model()


# class instance Iphone and Lawa class
#create objects of


# we can acess set_values methos using iphone instance
#iphone.set_values('1212sadasw145','Iphone-XR')



# now to see detailes of Iphone or Lawa we can call respective methods of their class
```

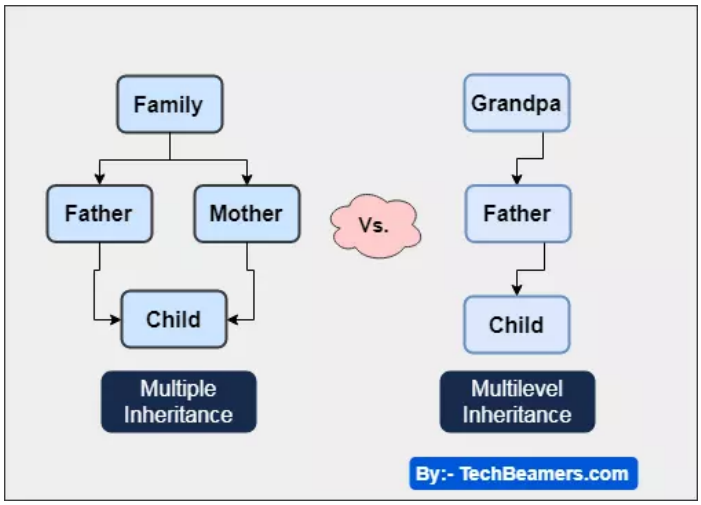
```
('1212sadasw145', 'Iphone-XR')
('asdasd23129bb23', 'Lawa-Agni')
```

Getter Setter Intro

Multiple Inheritance

Keep in mind there is always is a relationship between `Sub and Super Class`

- Say Lawa is a mobile
- Say Lawa and iphone both have some Price




```
##### Superclass-1 #####
#define class Mobile

    #declare private variable imei having value None

    #declare another private model having value None


    #define function set_values
                              # setting values
        #self.__imei = imei

        #self.__model = model


    #define function get_imei
```

```
                                # getting values
        #get imei value using self

    #define get_model function

        #get model value using self


##### Superclass-2 #####
#define class Price

    #declare private variable rs having None value


    #define function set_rs

        #assign rs value to private rs variable
        #self.__rs = rs


    #define get_rs function

        #get rs value using self


'''
Iphone Sub class is inheriting from Superclass Mobile and Price
'''
#### inherit from Superclass in sub class
#define class Iphone having Mobile,Price attributes

    #define function detail

        #return self.get_imei(),self.get_model()

#define class Lawa having Mobile,Price attributes

    #define function detail

        #return self.get_imei(),self.get_model()


# class instance
#create objects of Iphone and Lawa classes


# we can acess set_values methos using iphone instance
#iphone.set_values('1212sadasw145','Iphone-XR')

#lawa.set_values('asdasd23129bb23','Lawa-Agni')


#apply set_rs function by passing values
#iphone.set_rs(20000)

#lawa.set_rs(21000)


# now to see detailes of Iphone or Lawa we can call respective methods of their class
#print(iphone.detail())



#print(iphone.get_rs())
```

```
('1212sadasw145', 'Iphone-XR')
('asdasd23129bb23', 'Lawa-Agni')
20000
21000
```

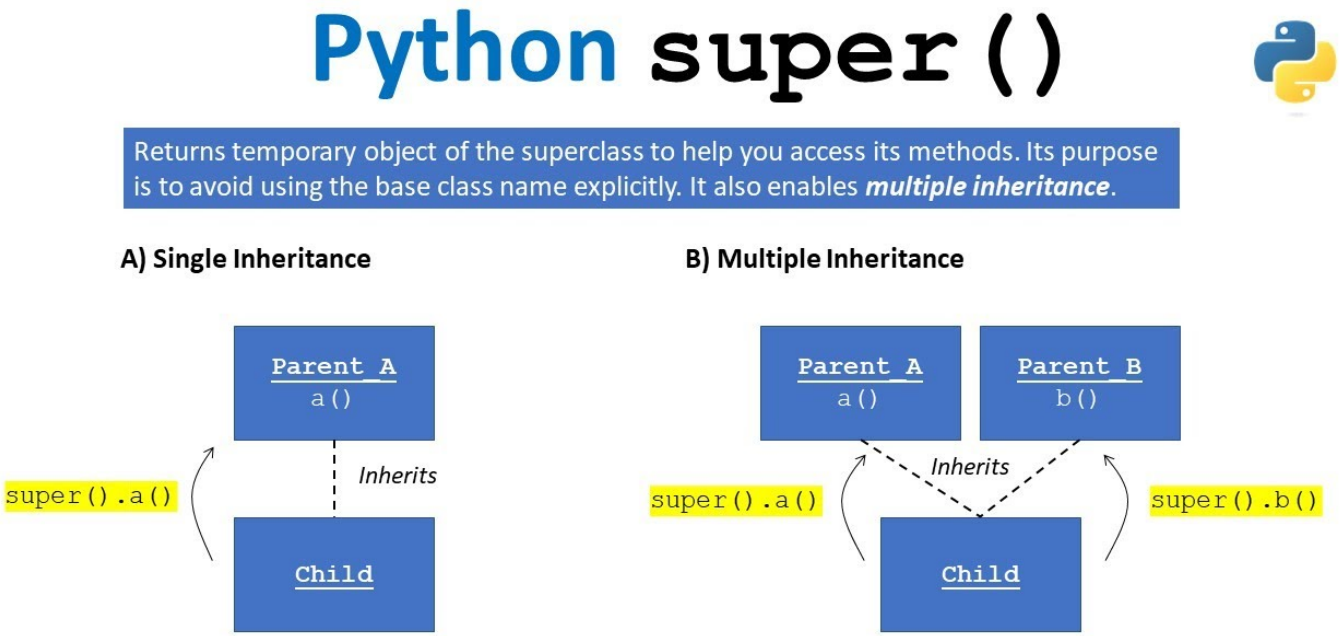
super()

How to use builtin Method Super() in classes

when instance of a class is created, `__init__` method is the firsts method which is called

NOTE:

`__init__` method ==> works as a constructor of a class




```
YouTubeVideo('MBbVq_FIYDA', width=600, height=300)
```


```
##### Papaji acting as a SuperClass ####
#define Papaji class

    #define init function

        #print('Parent __init__ is Executed')


#### Baitaji acting as a SubClass inheriting from parent class Papaji####
#create subclass Baitaji

    #define init function
```

```
        #print('Child __init__ is Executed')


#Object of Child class


```

```
 Child __init__ is Executed
```

See Subclass got Executed not Superclass i.e Constructor of Baitaji class in trigered

**What if you have a parameter inside SuperClass and you want to call that parameter which is in SuperClass(paretclass) from Child(Subclass)*

*`super() method` comes in rescue, see how ==> it helps to refer to Superclass Implicitly

- super() method return's proxy object that allows usr to refer to SuperClasss

See implementation

```
#define Papaji class

    #define init function

        #print('Parent __init__ is Executed')


#create subclass Baitaji

    #define init function

        #print('Child __init__ is Executed')

        '''
        will return proxy object of SuperClass
        calling __init__ method from SuperClass using super() function as ==> super().__init__()
        '''
        #apply super method
        #super().__init__(name='Papaji name')


#create Baitaji object


```

```
 Child __init__ is Executed
 Parent __init__ is Executed
```

See above output:

1. Child class's methos is active then
2. Using super(), we are abble to acess name--variable of SuperClass i.e. it is refering us to SuperClass

```
#define Papaji class

    #define init function

        #print('Parent __init__ is Executed')


#create subclass Baitaji

    #define init function

        #print('Child __init__ is Executed')

        '''
        will return proxy object of SuperClass
        calling __init__ method from SuperClass using super() function as ==> super().__init__()
        '''
        #apply super method
        #super().__init__(name='Papaji name')
```

```
#create object of child class


##### Mehod Resolution Order : can be acessed by class name or Child ####
#print('\nMehod Resolution Order is :\n',Baitaji.__mro__)
```

```
Child __init__ is Executed
Parent __init__ is Executed

Mehod Resolution Order is :
 (<class '__main__.Baitaji'>, <class '__main__.Papaji'>, <class 'object'>)
```

Rules based on Mehod Resolution Order works:

1. Subclass(here Baitaji) are always called first
    - then methods inside BaseClass are called

```
'''
MULTIPLE-INHERITANCE
Mehod Resolution Order ==> shows order in which methods are called inside
    Childclass or  Parentclass
'''
```

```
        '''
        will return proxy object of SuperClass
        calling __init__ method from SuperClass using super() function as ==> super().__init__()
        '''
```

```
##### Mehod Resolution Order : can be acessed by class name or Child ####
```

```
Child __init__ is Executed
Parent __init__ is Executed

Mehod Resolution Order is :
 (<class '__main__.Baitaji'>, <class '__main__.Papaji'>, <class '__main__.Papaji_1'>, <class 'object'>)
```

`Rule based on multiplt Inheritance:` see code above

1. Based on order of Inheritance

see above output

`Problem in Output:`

- Output of first Superclass can be seen only ==> by first super class means the Superclass being passed inside Baseclass in order say
    - `class Baitaji(Papaji_1,Papaji)` or `class Baitaji(Papaji,Papaji_1)`

As we have multiple inheritance in the code so we have to:

- Manually call all required `__init__` methods using Class Name

See code below

```
'''
MULTIPLE-INHERITANCE
Mehod Resolution Order ==> shows order in which methods are called inside
    Childclass or  Parentclass
```

```
'''
#define class Papaji

    #define function init

        #print('Parent __init__ is Executed',name)


#define Class Papaji_1

    #define init function

        # print('Parent_1 __init__ is Executed',name)


#class Baitaji(Papaji_1,Papaji):

    #define init function

        #print('Child __init__ is Executed')


        '''
        will return proxy object of SuperClass
            calling __init__ method from SuperClass using super() function as ==> super().__init__()

        Calling every __init__ method manually using class name as we have multiplt inheritance in our code so we cant use super()
            Here self will be required as a parameter
        '''
        #super().__init__(name='Papaji name')



#create object of Baitaji class


##### Method Resolution Order : can be acessed by class name or Child ####


     Child __init__ is Executed
     Parent_1 __init__ is Executed name_1
     Parent __init__ is Executed name

     Mehod Resolution Order is :
      (<class '__main__.Baitaji'>, <class '__main__.Papaji_1'>, <class '__main__.Papaji'>, <class 'object'>)
```

Composition

It allows to `delegate some responsibility from one class to another class`

- In Class Composition `one class acts like a Container` and `other acts like a content`

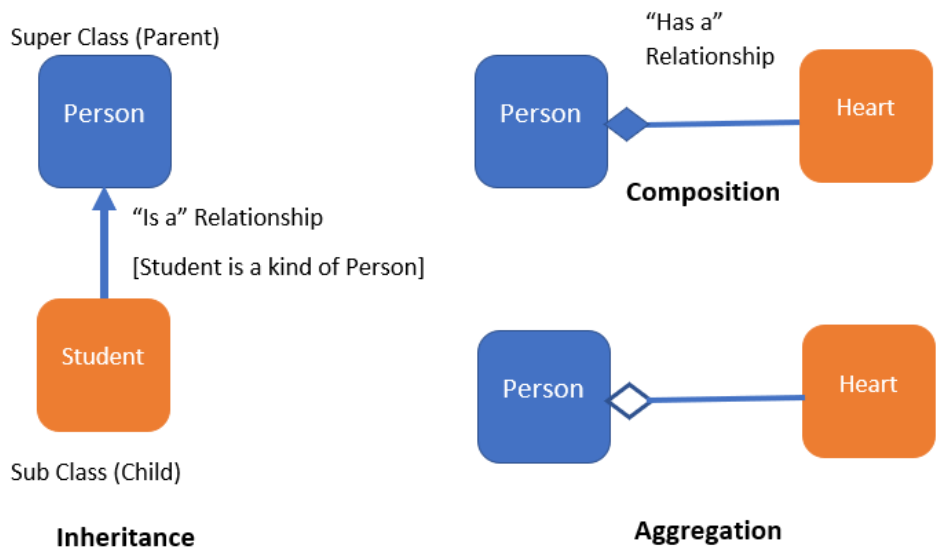
Composition represents part-of relationship

- When there is a composition between two Python classes, content object cannot exist without container object

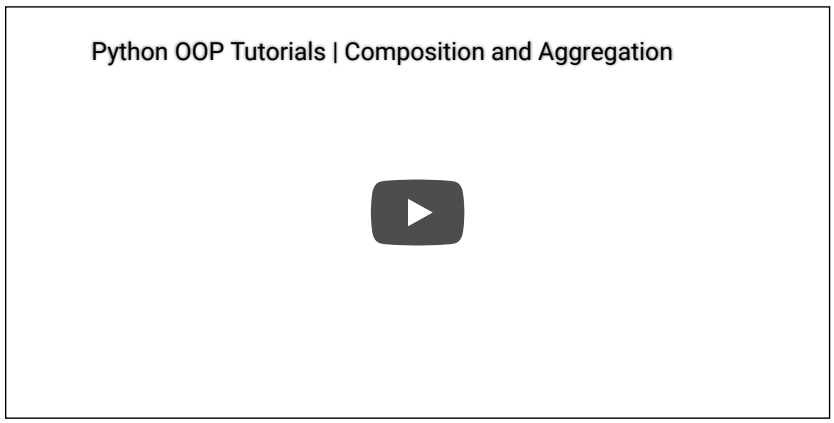
`When to use Composition??` ==> in general if you dont have is a relationship and you cannot apply Inheritance then we use Composition

`Composition` ==> it means deligating some responsibilities from one class to another class

UML Diagram




```
YouTubeVideo('mPFc3JHLnz8', width=600, height=300)
```




```
#define class Salary

    '''
    this constructure takes in arguments named as pay and bonus
    '''
    #define init function

        #assign pay value using self

        #assign bonus value to self

    #define anual_salary function

        #return (self.pay*12) + self.bonus


#define class Employee

    #define init function taking name,age,pay,bonus

        #assign name value using self

        #assign age value using self

        '''
        Making a salary object i.e. obj_salary using Salary class
            As Salary class takes 2 parameters like pay and bonus, which we are alrady providing
            using Employee class

        We are instantiating Salary Class inside Employ class in bellow line of code
```

```
        Here Employee class is acting as a ==> Container of Salary class
            Salary class is acting as an ==> Content of Employ class
        '''
        #self.obj_salary = Salary(pay,bonus)


    #define total_salary function

        '''
        using obj_salary member variable
        '''
        #return self.obj_salary.anual_salary()


# creating instance of Employee class



    Anual Salary is: 280000
```

Above anual salary is calculated using Salary class

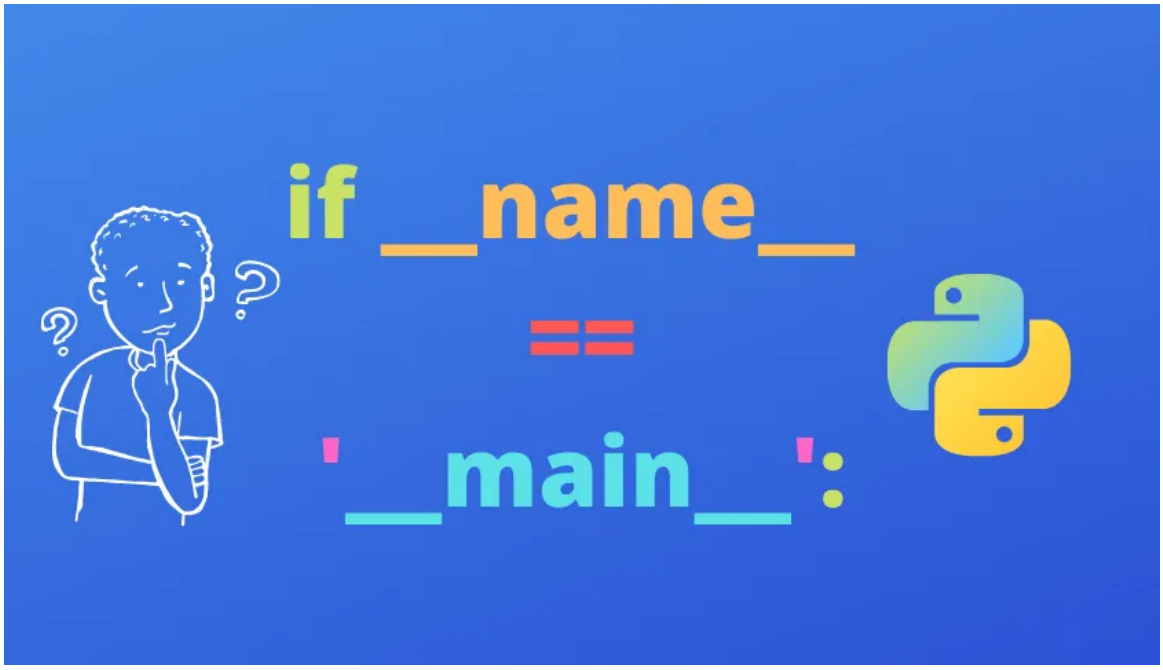
Employ class have Delegated some responsbility to Salary class which gives us annual salary of an Employ

`employee.total_salary()` ==> this is composition

Book and Chapter class can be another example of Composition

- As Book is not a Chapther but we can Deligate some responsibility from Book to Chapter class

if **name** == "**main**"



1. Where to use kework `__name__` ?
2. Why to use special if statement i.e. `if __name__ == "__main__"` ?

Some idea behind `if __name__ == "__main__"`

- `__name__` ==> a keyword
- `__name__` ==> built in variablevalue
- value of `__name__ is set to "__main__"` at runtime

Each module has a name and this name can be used to access instructions in a module

- This is especially useful in a given situation: `main block of a module is executed when it is imported for first time`

- But what do we do if we want block to run only if module was started as a stand-alone program, but not if it was imported from another module?

- We can do this by evaluating attribute `__name__` of module

Keep in mind:
`__name__` keyword value is either

- module name(say main.py) or
- `__main__`

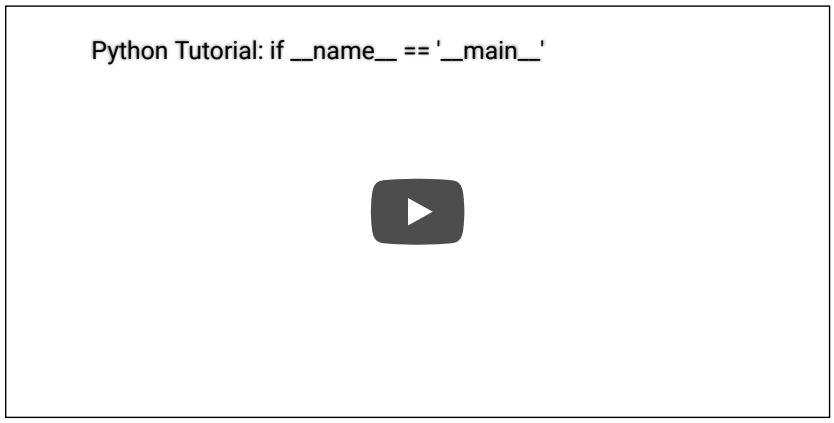
Question is:

1. In which condition `__name__` keword value becomes equall to module name??? ==> when we run code in main python file
2. In which condition `__name__` keword value becomes equall to `__main__` ??? ==> when we import file as a module in other file

We can say this condition ==> `if __name__ == "__main__"` is similar as main method in C++

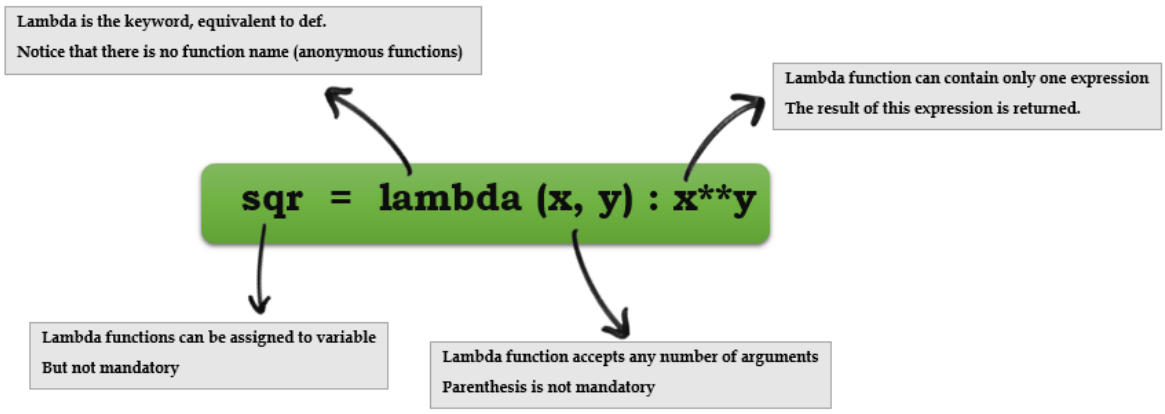
- main method is the main entrypoint of the program
- so can say above condition is performing same task

```
YouTubeVideo('sugvnHA7ElY', width=600, height=300)
```



`General Note` : Python is multi-paradigm language and supports Functional Programming

- lambda function is part of Functinal Programming paradigm
  - lambda function do not have any name
  - also called oneline function



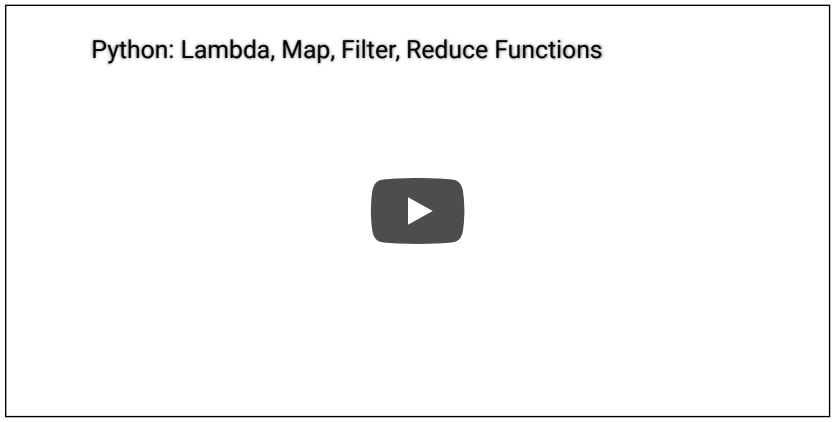
Lambda,Map,Reduce,Filter

You have these Question in mind:

1. What is a lambda function and it's use?
2. Application of lamda function with other function like Map,Filter,Reduce?

Come dive in with me.....

```
YouTubeVideo('cKlnR-CB3tk', width=600, height=300)
```




```
##### simple function #####
#define power function taking a number

    #return number powered to 2

#define add function taking two numbers

    #return addition of the numbers

#define sub function taking two numbers

    #return subtraction of the numbers


##### changing simple function to lambda function #####
#power_lamd = lambda number : number ** 2
                                    #not using return keyword
#do the same for other two functions

                                    # using "," as seperator


# using lambda function
#print(power_lamd(number=108))

#do the same for other two functions
```

```
11664
209
7
```

```
# using general function
```

```
11664
209
7
```

As above written general function are not that heavy in work so they can be written as, see below

```
##### simple function can be declared as single line function#####
#def power(number): return number * 2

#do the same for other two functions
```

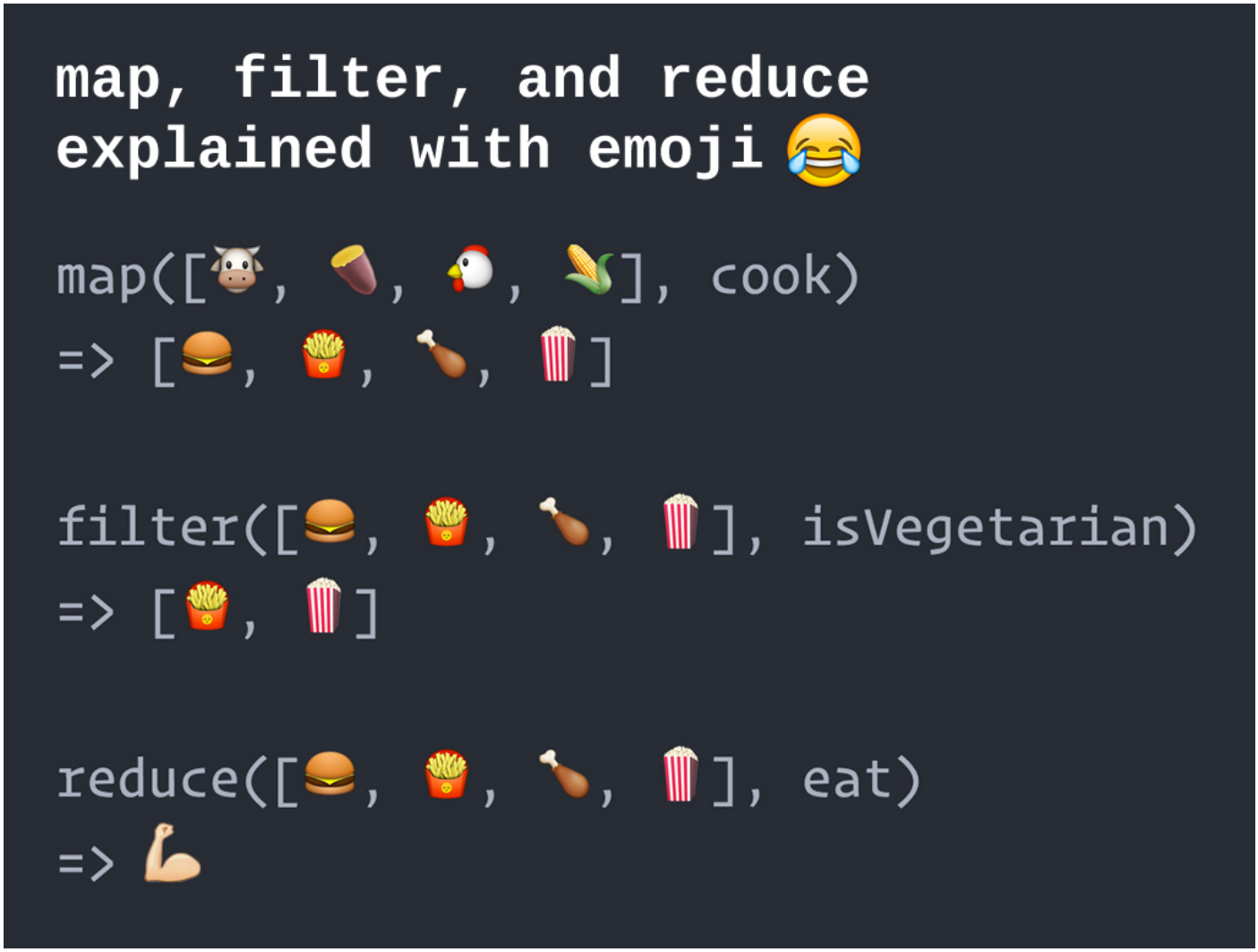
Why to use lambda function then if we can write general function like above

1. Lambda function are used generally with functions which take
    - function as an arguments or
    - return function as a result

In functional programming Function are first citizen, it means

- function can be passed as an normal argument

This is where Lambda function is used



Map Function

```
# using lambda function with other function like Map,Reduce, Filter
#create list haviny random numbers


#map function takes a (function and a iterator variable) i.e list, tuples or dict
#result = map(lambda x : x**2, dummy_list)
                                        # function is => "lambda x : x**2"  and iterator collection is => dummy_list
#print result

#print list of result
```

```
<map object at 0x7f4955ad3d50>
```

```
[10201, 11664, 12321, 14641, 100160064]
```

Above map is applying function "lambda x : x**2" to each element of list "dummy_list"

```
# adding two lists using map and lambda function
#create two lists having random numbers


#aaply lambda and map functions to add numbers from both the lists

#print list of result
```

```
[10302, 11772, 12432, 14762, 100170072]
```

Filter Function

It take function in firt argumen but condition is that `function must return boolian result`

```
# find only number divisible by 2
#consider output list of above code

#apply filter function,filter(lambda x : x%2==0, list_addition_result)

#print list of result
```

```
[10302, 11772, 12432, 14762, 100170072]
```

```
# use filter to find number greater then 12432 in list list_addition_result
```

```
[14762, 100170072]
```

Reduce Function

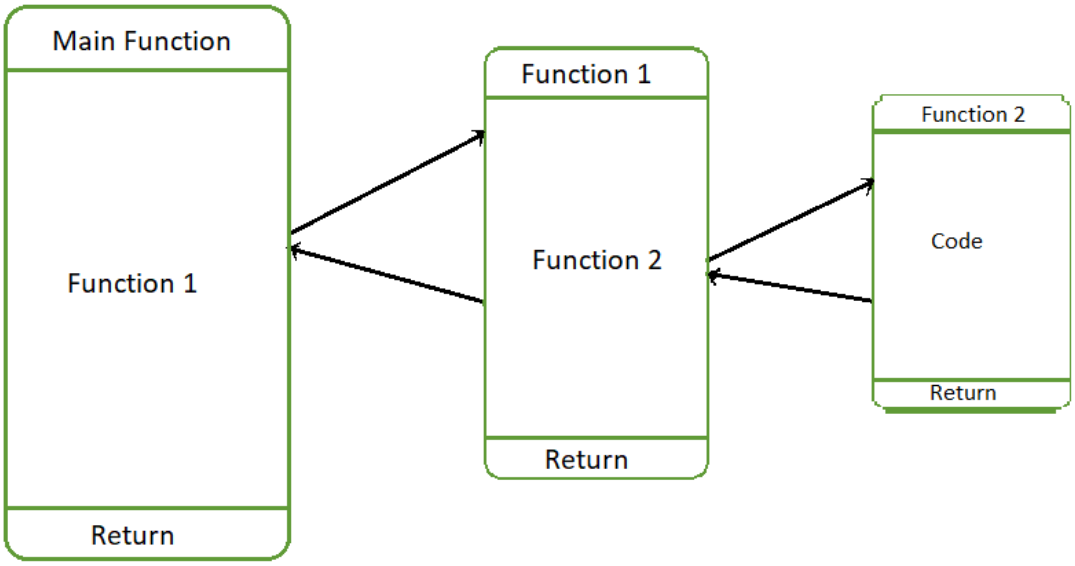
https://www.geeksforgeeks.org/reduce-in-python/

```
# require module to use reduce function
                                  #from functool module we require reduce function

#create a dummy list

#apply reduce function by passing some operation
#reduce_list_result = reduce(lambda x,y : x-y,dummy_list)
```

```
-34
```

Nested Functions

https://www.geeksforgeeks.org/python-inner-functions/#:~:text=A%20function%20which%20is%20defined,is%20also%20known%20as%20Encapsulation%20.



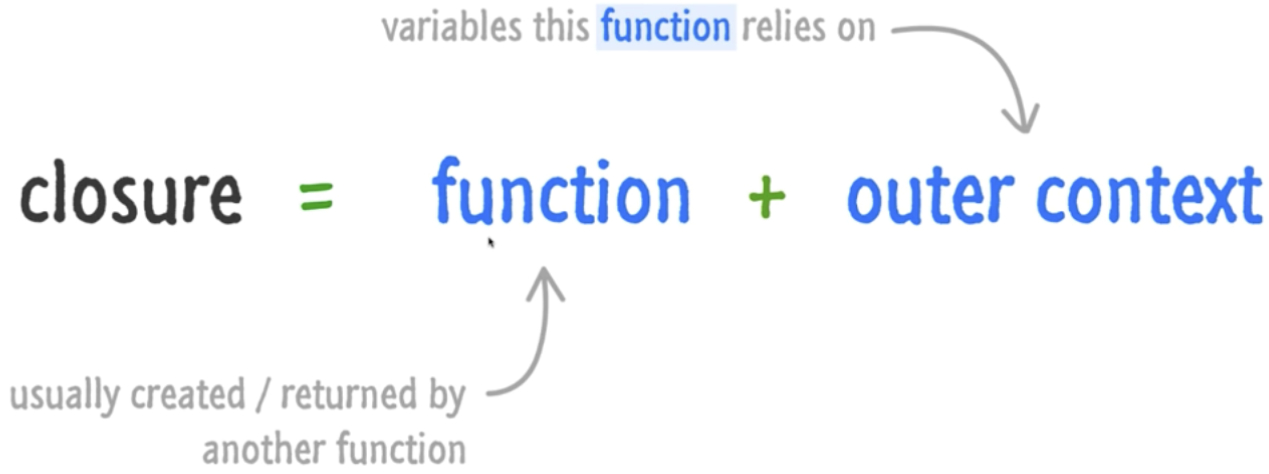

```
#### Function nesting Example ####
#create an outer function taking a message

    #create an inner function

        #print message

    #inner_local_Function_of_outerFunction()
                      # acessing inner function in scope of outer function

#calling outer function


     Welcome to CloudyML, I am your Python guide Mukesh Manral


#### more complex example #####
'''
Nested function to remove second last element of a taken list
'''
#create an outer function taking a dummy list

  #create an inner function with a dummy list 1

      #return dummy_list_1[len(dummy_lis)-1]

      #dummy_list.remove(extract_list_last_element(dummy_list))

    #return the dummy list


#giving dummy list

#printing these 4 times as we have given 5 (n-1)integers
#print(removee(dummy_list = given_dummy_list))



     [101, 108, 111, 121]
     [101, 108, 111]
     [101, 108]
     [101]
```

Closures

- Closure is a function object that remembers values which are declared outside the functions, even when that is not in memory

- Down code `message` variable can be seen

- Closure lexical environments are stored in `property __closure__ of a function`



Use of Closures:

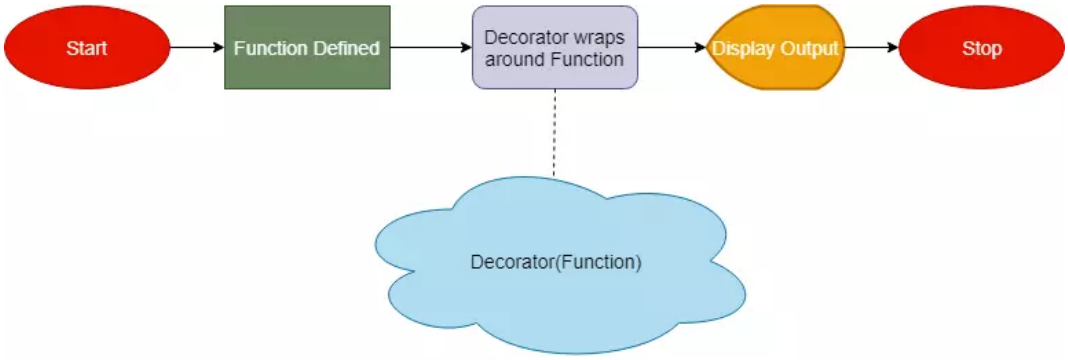
1. Closures can be used in place of classes
   - Only classes which have fewer methods generally one method inside them
2. Closures are used moslty in case of Decorators.
3. Closures can be more effective then Classes sometime.

Return inner function in scope of Outer function without parenthesis i.e. ()

```
#### Function nesting Example ####
#creating outer function

  #creating inner function

        #printing messege here

    #return the inner function here
                                      # acessing inner function in scope of outer function  <<===== Focus

#calling outer function

#del outer_enclosing_Function

                                    #calling inner function ==> will work even outer function is deleted, because it stores state

     Welcome to CloudyMl, I am your Python guide Mukesh Manral
```

Decorator(@deco)

It wrap's a function and modify its behaviour in required way's, without having to directly change source code of actual function i.e. function being decorated



Lets se by writing a function-decorator that can be used to track run time of a functions

1. How to declare decorator?
2. How to use decorator?

```
YouTubeVideo('r7Dtus7N4pI', width=600, height=300)
```


```
"""
defining a function
"""


"""
What if you want to print some msg after and before above print statement but with a Constraint:
  `You cannot change code of above function` ==> Decorator comes in rescue
1. Declare a decorator function which take func which is a function as an argument
2. Declare another function inside deco_func whic is a wrapper_func
3. wrapper_func takes no arguments
4. inside this wrapper_func which is a function we are calling function which was passed as an argument to deco_function
5. returning wrapper_func without parenthesis i.e. () in scope of dec_function
  5.1: we can return inner function without parenthesis that make them Closure
"""
# one argument is given i.e func ==> a function
#creating a function here

  #creating a inner function here

    #func()

                # closure

"""
Now we can pass above welcome_msg function as an argument in place in func to deco_func function
  and can modefyie output of welcome_msg function as required Lets seee how
"""

##########################################################################################################################################################
##########################################################################################################################################################

#creating a function here

  #creating a inner function here
```

```
    #before message

    #func

    #after message
                                 #after message
    #return inner function
                                # closure


#creating function here

  #print("enter a msg ")


"""
Callling welcome_msg fucntion with decorator
1. Providing deco_function welcome_msg function as an argument
2. As decorator is returning inner function so variable welcome will contain inner function
3. As now inner function is inside welcome variable and see inner function i.e. wrapper_func() dont take any argument so
4. So it can be called as welcome()
"""
```

```
     Hi there
     Welcome to cloudyML
     This is your Instructure Mukesh Manral
```

But But But Python provide a simple way to declare this kind of notation

```
"""
welcome = deco_func(welcome_msg)
welcome()
"""


#creating function here

  #create inner function

    #before message

    #func

    #print #after message
                               #after message
  #return inner function
                               # closure


"""
this notation ====> @deco_func  is same as
this notaion =====>
                    welcome = deco_func(welcome_msg)
                    welcome()
"""

"""
function on which we want to apply decorator use:
  @name_of_decorator --- to apply on fucntion
"""
#create decorator

#creating function

  #printing message


# now directly use
```

```
     Hi there
     Welcome to cloudyML
```

```
This is your Instructure Mukesh Manral
```

Remember def of Decorators:

It wrap's a function and modify its behaviour in required way's, without having to directly change source code of actual function

see above Output now how we have changes output of function `welcome_msg()` without changing it's code... clear i guess... but but but.....

Can We use more then one Decorator with same function??

Lets see by doing..

1. Will declare two decorator functions
2. Will change there output a bit

```
#creating function

  #creating inner function

    # print before message

    #func

    #print after message

    #return inner function
                         # closure

#creating function

  #creating inner function

    #print before message

    #func

    #print after message

    #return the inner function
                                  # closure

# will execute first

# will execute second


#creating function

  #print


#call the function
```

```
Hi there from Rahul side
Hi there from Mukesh Manral side
Welcome to cloudyML
This is your Instructure Mukesh Manral
This is your Instructure Rahul
```

See how @deco_func_2 have wrapped main function i.e. welcome_msg() output first, as it was given first in order and so on..

Always keep in head the order of decorator

```
### another long way is like this

#creating function

  #create the inner function

    #print before message
```

'@deco_func_2' is not an allowed annotation – allowed values include [@param, @title, @markdown].

'@deco_func_1' is not an allowed annotation – allowed values include [@param, @title, @markdown].

```
      #func

      #print after message

    #return inner function
                  # closure

  #creating function

    #create the inner function

      #print before message

      #func

      #print after message

    #return the inner function
                  # closure

  #@deco_func_2 # will execute first --- but for now commented
  #@deco_func_1 # will execute second --- but for now commented
  #create the function

    #print the message


  # deco 2 is wraping deco 1, reverserd can be done
```

```
Hi there from Rahul side
Hi there from Mukesh Manral side
Welcome to cloudyML
This is your Instructure Mukesh Manral
This is your Instructure Rahul
```

Now you can claim that you have basic intution about Decorators...
Lets make you concept of Decorators bit more dimond solid

1. This time our function will take 2 arguments
2. Based on our function we will modefyie our wrapper_func
3. This time wrapper function will print somthing too.

```
#Create the function

  #create the inner function

    #print
    #print('Mulitiply',x_1, 'and', x_2)

    #if x_1 == 0 or x_2 == 0:

      #print(gives 0)


    #return x_1 * x_2

  #return inner function


"""
This time we are giving some arguments to our say main function
"""
#create decorator

#create a function with num1,num2

  #return 2 nos
```

```
#calling the function


     Mulitiply 0 and 0
     Multiplying with 0 gives 0
```

What we have covered in Decorator so far is as:

1. We have created Decorator function for function which take No-Argumens
2. We have created Decorator function for function which take some-Argumens say two in above case
    - we have used these arguments inside wrapper function

In real life you have to write most of time `Generic Decorator Function` ==> Deco func which can be used with multiple function

`Writing a Generic Decorator Function which tells execution time of any Function`

Will use `*args` and `**kwargs` to make wrapper function independent of arguments which will be given by `func` function

Will use inbuilt time module

```
#import time


# general time function which can be used with any function
#creating function

  #creating inner function with *args,**kwargs

    #current_time

    #print the start time

    #as we dont know what and homany will be input

    #current time after some time i.e. after execution of function

    #print the end time

    #total timke taken

                                # this or other function can return some values so using this
  #return inner func



# decorating function now

#define a function

  #initialize 0 with a variable

  #iterate over numbers

    #num=num+1

    #return num


#call the function


     start_time is: 1643910794.306848
     end_time is: 1643910794.3080423
     Time taken in function execution: 0.0011942386627197266
     210
```

Congratulations!!! You've done it.

In this assignment you learned about OOPs concept in Python.

Keep practising!!

Do fill the feedback form given below:

[Feedback form]

See you then!!