OOP (OBJECT ORIENTED PROGRAMMING)

way of thinking about and organizing code for maximum reusability

group of functions

function : group of instructions

everything in python is object --> copy of class

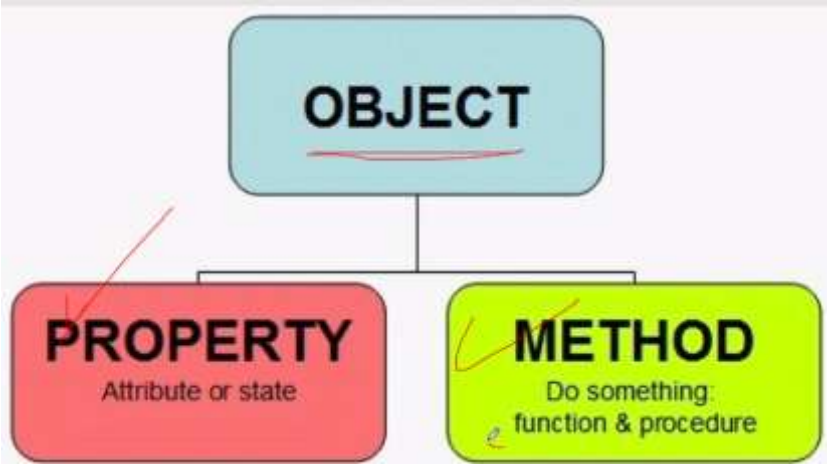
class consist of

1. attributes (state)

specifications - property

2. methods

functions



how to create class

class attributes VS instance attributes

class attributes: attribute of the main class

instance attributes: attribute of the object

In [6]:

```
class BMW:
    #class attributes
    color = 'red'
    MODEL = 2022

    # methods

    def open_door():
        print("open door")
    def close_door():
        print("close door")
# opbject  = copy
copy = BMW()

print("color of BMW car before edit =  ",copy.color)

print("#####################################################")

# instance attribute ( object attribute) (attribute related to object only)
copy.color = "black"

print("color of BMW car before edit =  ",copy.color)

copy2 = BMW()

print("#####################################################")
## Guess the output

print("color of BMW car in object copy2 =  ",copy2.color)
```

```
color of BMW car before edit =   red
#####################################################
color of BMW car before edit =   black
#####################################################
color of BMW car in object copy2 =   red
```

examples of methods

In [7]:

```
class BMW:
    #attributes
    color = 'red'
    MODEL = 2022

    # methods

    def open_door():
        print("open door")
    def close_door():
        print("close door")
# opbject  = x
x = BMW()

x.open_door() # error
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Input In [7], in <cell line: 15>()
     12 # opbject  = x
     13 x = BMW()
---> 15 x.open_door()

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

In [8]:

```
class BMW:
    #attributes
    color = 'red'
    MODEL = 2022

    # methods

    def open_door(self):
        print("open door")
    def close_door(self):
        print("close door")
# opbject  = x
x = BMW()

x.open_door()
```

```
open door
```

proof that object is main argument of mthods you have (replace self)

object passing to method functions and replace self by python

In [9]:

```
class amit:
    def testt(self):
        print(self)
        print("--------------------")
x = amit() # passing x to self
x.testt() # print sotred value of self
print(x) # print value of x when it pass to self
```

```
<__main__.amit object at 0x0000025C28CE7E50>
--------------------
<__main__.amit object at 0x0000025C28CE7E50>
```

Encapsulation

1. process which are protective layer ensure access to data is not possible by any code defined outside the class

2. this mean Encapsulation provides security by hiding the data from the outside world.

private data inside class :they can be accessed only

In [10]:

```
class Rectangle:
    __length = 5 # private attribute
    __width = 7 # private attribute
    height = 4  # public attributes
r1 = Rectangle()
print(r1.height)
#print(r1.width) # error
#print(r1.length) # error
```

```
4
```

In [11]:

```
class Rectangle:
    __length = 5 # private attribute
    __width = 7 # private attribute
    height = 4  # public attributes
    def private_variables(self):
        self.__length = 5 # private attribute
        self.__width = 7 # private attribute
        print("length inside class = ",self.__length)
        print("length inside class = ",self.__width)
r1 = Rectangle()
print(r1.height)
r1.private_variables()
#print(r1.width) # error
#print(r1.length) # error
```

```
4
length inside class =  5
length inside class =  7
```

constructor

special type of method

method btsht8l awl ma y7sal passing ll object to self

In [12]:

```
class studnet:

    #def student_info(self,name,age):
    #    self.name = name
    #    self.age = age
    #    print('i am {} and i am {} years old'.format(name,age))

    #constructor method
    def __init__(self):
        print('i am constructor method')

s1 = studnet()
#s1.student_info('amit learning',30)
#s1.student_info("amit",30)
```

```
i am constructor method
```

replace student_info method to construction method

In [13]:

```
class studnet:
    def __init__(self,name,age):
        self.name = name
        self.age = age
        print('i am {} and i am {} years old'.format(name,age))
#constructor method
'''
    def __init__(slef):
        print('i am constructor method')
     '''
s1 = studnet("amit learning",30)
#s1.student_info('amit learning',30)
#s1.student_info("amit",30)
```

```
i am amit learning and i am 30 years old
```

example 1 OOP

1) A3ML barnamg b ast5dam oop bya5od mnk el name bta3 sha5s w bta5od mnk daraga w btdyfha fy list gwa el oop w t7sb el average bta3 drgat el student

In [14]:

```
class student:
    #constructor
    def __init__(self,name):
        self.name = name
        self.marks = []
        print("wlecome {} in the school".format(name))
    def add_marks(self,mark):
        self.marks.append(mark)
    def avg(self):
        print("average of marks = ",sum(self.marks)/len(self.marks))
s1 = student("amit")
print("-"*10)
print("number of marks in list before adding = ",s1.marks)
print("-"*10)
s1.add_marks(40)
print("number of marks in list after adding = ",s1.marks)
print("-"*10)
s1.add_marks(60)
print("number of marks in list after adding = ",s1.marks)
print("-"*10)
s1.avg()
```

```
wlecome amit in the school
----------
number of marks in list before adding =  []
----------
number of marks in list after adding =  [40]
----------
number of marks in list after adding =  [40, 60]
----------
average of marks =  50.0
```

OOP example 2

create calculator that add subtract multiply divid and check even or odd numbers using oop

In [15]:

```
class calculator:
    def __init__(self,first_number,second_number):
        self.first_number = first_number
        self.second_number = second_number
    def add(self):
        print("summation = ",self.first_number + self.second_number)
    def subtract(self):
        print("subtraction = ",self.first_number - self.second_number)
    def multiplication(self):
        print("multiplication = ",self.first_number * self.second_number)
    def division(self):
        print("division = ",self.first_number / self.second_number)

n1 = int(input("enter your first number: "))
n2 = int(input("enter your second number: "))
calc1 = calculator(n1,n2)

calc1.add()
calc1.subtract()
calc1.multiplication()
calc1.division()
```

```
enter your first number: 5
enter your second number: 4
summation =  9
subtraction =  1
multiplication =  20
division =  1.25
```

inheritance concept (وراثه)

class btwrs haga mn class tnya el attributes bt3tha aw el methods

without using inheritance

3ayz a3ml 2 calculators by3mlw add w multiply bs tany calculator bt3ml add w multiply w power (rakam power rakam)

In [16]:

```
class calculator1:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def sum(self):
        print("sum = ",self.x+self.y)
    def multiply(self):
        print("sum = ",self.x*self.y)

class calculator2():
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def sum(self):
        print("sum = ",self.x+self.y)
    def multiply(self):
        print("sum = ",self.x*self.y)
    def power(self):
        print(f"{self.x}^{self.y}={pow(self.x,self.y)}")

x = calculator2(2,3)
x.power()
```

```
2^3=8
```

using inheritance

In [17]:

```
class calculator1:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def sum(self):
        print("sum = ",self.x+self.y)
    def multiply(self):
        print("sum = ",self.x*self.y)
# inheritance
class calculator2(calculator1):
    #delete def __init__(self,x,y):
    #delete     self.x = x
    #delete     self.y = y
    #delete def sum(self):
    #delete     print("sum = ",self.x+self.y)
    #delete def multiply(self):
    #delete     print("sum = ",self.x*self.y)
    def power(self):
        print(f"{self.x}^{self.y}={pow(self.x,self.y)}")

x = calculator2(2,3)
x.sum()
```

```
sum =  5
```

inherting class name: (el class elly btwrs mn class tnya) (calculator2)

1.child class

2.Derived class

3.subclass

inherted class name:(el class elly btwrs mnha) (calculator1)

1.parent class

2.base class

3.super class

```
class calculator2(calculator1):
    #def   init  (self,x,y):
```

constructor inheritance (super)

we use super() function to call methods in parent class

lw ana 3ayz awres haga mo3ayana mn class elly bawrs mnha

In [18]:

```
class Animals(object):
    def __init__(self,name):
        self.name = name
class Dog(Animals):
    def __init__(self,name):
        super(Dog,self).__init__(name)
        self.food = "meat"
d = Dog("rex")
print(d.name)
print(d.food)
```

```
rex
meat
```

multiple inheritance

(ezay ana mmkn awrs mn aktr mn source)

In [19]:

```
class A:
    def dothis(self):
        print("i am in A")
class B(A):
    pass
class C:
    def dothis(self):
        print("i am in c")
class D(B,C):
    pass
s = D()
s.dothis()
# Method Resolution Order (way a programming language resolves a method or attribute.)
print(D.mro())
```

```
i am in A
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.A'>, <class
'__main__.C'>, <class 'object'>]
```

In [20]:

```
class A:
    def dothis(self):
        print("i am in A")
class B(A):
    pass
class C:
    def dothis(self):
        print("i am in c")
class D(C,B):
    pass
s = D()
s.dothis()
# Method Resolution Order (way a programming language resolves a method or attribute.)
print(D.mro())
```

```
i am in c
[<class '__main__.D'>, <class '__main__.C'>, <class '__main__.B'>, <class
'__main__.A'>, <class 'object'>]
```

polymorphism (many shapes)

two classes with same interface (same method name but diffrent in function)

In [21]:

```
# len here is method in string class
print(len("hello"))
# len here is method in tuple class
print(len(("hello","hello","hello")))
# len here is method in list class
print(len(["hello","hello","hello"]))
```

```
5
3
3
```

conclusion

1. everything is object (object = copy of class)

2. classes = attributes + methods

3. self => bthgz makan ll object 3shan el object hyt3mlw pass ll class bt3tk

4. constructor (method awl ma ta5od object mn class el contructor method bttnfz

5. inhertance(class btwrs mn class tnya badal ma3od akrr nfs el attributes w methods

6. polymorphism(2 class fyhom nfs el method {nfs asm} bs kol method lyha function mo3ayana)

7. construction inheritance (super)

btwrs attripute mo3ayana mn class bta3k b ast5dam klmt super

8. multiple inheritance

bawrs mn kaza class w 3shan a3raf el sequence elly bawrs byh bast5dm

Method Resolution Order (way a programming language resolves a method or attribute.)