In [ ]:
```
#Agenda of the Today:
         1. Object Oriented Programming in Python
```

In [ ]:
```
#Introduction about Oops?
    ---In 1960's Oops was initiated by alan kan
    --- Initially a language called simula which is the first programming
        with object oriented features.
    --1990's is was avaiable in market with help of c++
    --after that it was adopted by many programming languages.

#Where we can use oops mostly:
   1. Real-time systems
   2. Artificial Intelligence
   3. Expert systems
   4. client-server systems
   5. Object-Oriented Databases, and etc.

 #What is the Object Oriented Programming?

---Its a different method of structuring a software program by bundling
 the properties and behaviors into individual objects.
      (or)
-- Its a programming paradigm that deals with classes and objects.

--Its used to structure a software program into simple,resuable pieces of code.

#Examples of oops:
  1. java
  2. javascript
  3. C++
  4. python
```

In [ ]:
```
#why use choose oops than pop?
  Oops vs pop?
#Key Points:

  1. pop- is done by functions
  2. Oops is done by divides the program into objects.
```

In [ ]:
```
#Contents of oops?
  1. class
  2. objects
  3. method
  4. Inheritance
  5. Encapsulation
  6. Polymorphism
  7. Data Abstraction
```

In [ ]:
```
#1. class?
  What is class?
    - A class is a collection of objects
        or
     A class is a blueprint of the object.
#Note: its makes the code as more managable.
```

In [ ]:
```
#How to create classes?
#syntax:
class class_name:
    #class body
```

In [ ]:
```
#ex:
student-object-class-csea
 properties-name,rollnum,emaild,phone num
 behaviour-topper,talk-active,silent,backbencher
```

In [3]:
```
#Class define
class car:
    pass
```

In [ ]:
```
#2. Object and Object Instantiation:
  what is object ?
    - An object (instance) is an instatiation of a class.
#Note: when the class is defined only the description or blueprint of
#a object is defined , here there is no memory allocation.
```

In [ ]:
```
# what is instantiation?
  -its creating a new object/instance of a class.
```

In [ ]:
```
#How to create objects?
 #Syntax:
     variablename = classname()
     obj = car()
```

In [ ]:
```
#Constructor in oops:
- Constructor in python is a special method which is
used to initialize the members of class during run-time
when an object is created.

we define class constructor always represented by double underscore "__"
    __init__().

#Note: Every class must have a constructor, even if you dont create
 a constructor explicitly it will create a default constuctor by itself.
```

In [7]:
```
#Ex:
class Myclass:
    sum = 0     #class variable
    def __init__(self,a,b):       #instance method or constructor method
        self.sum=a+b     #self keyword is used to access the members of class.
                         #a,b - instance variables
    def printsum(self):     #class method
        print("Sum of a and b is",self.sum)
#creating objects for above class
ob = Myclass(10,20)
ob.printsum()

#Note: constructor will never return values.
```

```
Sum of a and b is 30
```

In [ ]:
```
#Inheritance:
 - Inheritance is refers
    A class which inherts the properties of another class is called
```

```
      Inheritance.
                Properties
    Parent(class)---------->Child(class)
#types of inheritance:
    1. single
    2. Multilevel
    3. Multiple
```

In [13]:
```
#Single-Level Inhertance:
class arthimatic:       #parent class
    a= 10                           #parent class variable
    b = 20
    def add(self):            #parent class method
        sum = self.a+self.b
        print("sum of a and b is",sum)
class addition(arthimatic):       #child class
    c = 50
    d = 10                      #child class variable and method
    def sub(self):
        sub = self.c-self.d
        print("substraction of c and d is",sub)
ob=addition()    #child class object
print(ob.add())
print(ob.sub())
print(ob.a)
print(ob.b)
print(ob.c)
print(ob.d)
```

```
sum of a and b is 30
None
substraction of c and d is 40
None
10
20
50
10
```

In [ ]:
```
#Multi-Level Inheritance:
parent1class--->parent2(child1)--->child2class

In Multilevel - one or more class will act as a parent or base class.
```

In [17]:
```
#example:   (one or more parent classes)
class addition:             #parent class 1
    a = 10
    b = 20
    def add(self):
        sum = self.a+self.b
        print("sum of a and b is",sum)
class substraction(addition):   #parent class 2 or child class1
    def sub(self):
        sub = self.b-self.a
        print("subtraction of a and b is",sub)
class multiplication(substraction):     #child class2
    def mul(self):
        multi = self.a * self.b
        print("multiplication of a and b is",multi)
ob=multiplication()
ob.add()
ob.sub()
```

```
ob.mul()
ob.a
```

```
sum of a and b is 30
subtraction of a and b is 10
multiplication of a and b is 200
```

Out[17]: 10

In [ ]:
```
#Multiple Inheritance:
A class which is inherits the properties of multiple parent classes
is called mulitple inheritance:
```

In [19]:
```
#example:  (one child class and more than one parent classes)
class Addition:
    a = 10
    b = 20
    def add(self):
        sum = self.a+self.b
        print("sum of a and b is ",sum)
class substraction():
    c = 50
    d = 10
    def sub(self):
        sub = self.c-self.d
        print("substraction of c and d is",sub)
class multiplication(Addition,substraction):  #multiple inheritance
    def mul(self):
        multi = self.a * self.c
        print("multiplication of a and c is",multi)
ob= multiplication()
ob.add()
ob.sub()
ob.mul()
```

```
sum of a and b is  30
substraction of c and d is 40
multiplication of a and c is 500
```

In [ ]:
```
#Encapsulation:
       Its way to ensure security , simply it hides data from the access of outsiders.
-- for making encpasulation
    we can declare the methods or the attributes as "protected" by using
    a single underscore(_) before thier names.
     such as
        self._name or
        def _methodname()
#Note: By using some tricks there may chance of accessing the attributes and
       methods of class data (protected)
     # for preventing the access of attributes/methods from outsiders the scope of a cl
        you can make them as "private"
```

In [25]:
```
#Example:
class car:
    def __init__(self,name,mileage):
        self._name= name           #protected variable
        self.mileage = mileage
    def description(self):
        print(self._name,self.mileage)

ob = car("BMW 11-series",60)
```

```
print(ob.description())

#accesing the protected directly from outside

print(ob._name)
print(ob.mileage)
```

```
BMW 11-series 60
None
BMW 11-series
60
```

In [29]:
```
#make class variabels as private:
class car:
    def __init__(self,name,mileage):
        self.__name=name   #private variable
        self.mileage   = mileage
    def description(self):
        print(self.__name,self.mileage)
ob = car("BMW-11 Series",60)
print(ob.description())
print(ob.mileage)
print(ob.__name)
```

```
BMW-11 Series 60
None
60
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-29-5d5ecff2d0a5> in <module>
      9 print(ob.description())
     10 print(ob.mileage)
---> 11 print(ob.__name)

AttributeError: 'car' object has no attribute '__name'
```

In [ ]: