- ---
- ----------Today's Topic is :
- 1 . Class
- 2 . Object
- 3 . State
- 4 . Main aspects of OOPs
- 5 . Defining a Class
- 6 . Creating an Object
- 7 . Instance object and Class object
- 8 . init() method
- 9 . Attributes in class
- 10 . intance method creation and access
- 11 . Class object variable creation and access
- 12 . 4-Pillers-Oops
- 13 . other concepts
- ---
- Class
- Object
- State
- Main aspects of OOPs/4 Pillars of OOPs
- Defining a Class
- Creating an Object
- Instance object and Class object
  - class object
  - instance object
- ----------#
- init() method
- Acha Bahi ab deko working :
- ---------- Attributes in class ----------
  - Note:
  - Example:
- Note:
  - we have 4 types of methods/function in python
- ----------#
- intance method creation and access
  - Creation of instion ways:
  - Access of instion ways:

- ----------#
- Class object variable creation and access
  - Creation of class object variable ways:
- ----------
- ----------
- ----------4-Pillers-Oops----------#
  - other concepts
- ----------
- => act of combining properties and methods related to same entity/object.
- lets Example
- point to understand
- Note:
- ----------
- in python termnology
- ----------

----------Today's Topic is :

1 . Class

2 . Object

3 . State

4 . Main aspects of OOPs

5 . Defining a Class

6 . Creating an Object

7 . Instance object and Class object

8 . init() method

9 . Attributes in class

10 . intance method creation and access

11 . Class object variable creation and access

12 . 4-Pillers-Oops

```
Encapsulation
    Inheritance
    Polymorphism
    Abstraction
```

13 . other concepts

Class

```
=> A Class is Common Noun(common name) that represents a group of objects.
=> A Class is Collection of Objects.
```

```
=> like[tree, car, animal, human, etc]
=> A Class is a blueprint(template) for the object.
=> A Class is a user-defined data type.
=> A Class is a logical entity.
=> A Class is a classification of objects.
=> A Class is a group of objects that have common properties.
=> A Class is a community Name of objects.
```

common noun: [doctor, teacher, student, car, tree, animal, human, etc]

Object

```
=> An Object is a Proper Noun(Specific name) that represents a single entity.
=> An Object is a real-world entity that has state and behavior.
=> An Object is an instance of a class.
=> An Object is a collection of data and methods.
=> An Object is a container that contains data and methods.
```

proper noun: [doctor ali, teacher ahmed, student asad, car honda, tree mango, animal lion, human ali, etc]

Example: - Class: Pen - Object: Blue Pen, Red Pen, Green Pen, etc. - Class: Chair - Object: Wooden Chair, Plastic Chair, Iron Chair, etc. - Class: Table - Object: Wooden Table, Plastic Table, Iron Table, etc. - Class: Computer - Object: Dell Computer, HP Computer, Lenovo Computer, etc. - Class: Mobile - Object: Samsung Mobile, Nokia Mobile, iPhone Mobile, etc. - etc.

- In the world each and everything is an object.
    - Class: Human
    - Object: Ali, Ahmed, Asad, etc.
    - Class: Animal
    - Object: Dog, Cat, Lion, etc.
    - Class: Plant
    - Object: Mango Tree, Apple Tree, etc.
    - etc.
    - In programming, an object is a container that contains data and methods.
    - An object is a collection of data and methods.
    - An object is an instance of a class.

State

```
=> set of properties,value,attributes,variables,fields,characteristics,features,
that are stored in an object.
1. methods: write, refill, cap_on, cap_off
2. attributes: color, ink, brand
```

Main aspects of OOPs/4 Pillars of OOPs

```
1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction
```

python does not support Data Hiding concept.

Defining a Class

```
=> A class is defined using the class keyword.
=> A class is defined using the class keyword followed by the class name.
=> A class name should start with an uppercase letter.
```

Syntax: class ClassName: # class body

```
# Example:
class Pen:      # class definition
    pass
```

Creating an Object

```
=> An object is created using the class name followed by parentheses.
```

Syntax: object_name = ClassName()

```
# Example:
p1 = Pen()

# but when
pen()       # basicallly this is a function calling but pen is class name  atually
when we call class name with () then it will create an object of this class and
then we store this object in a variable and access the properties and methods of
this class.

# we can make multiple objects of a single class with different names and its
values.

p2 = Pen()
p3 = Pen()
```

Instance object and Class object

class object

```
=> jasy hi app ne koi class define to smjlo usi waqt us class ka object bna jata hai
wo object class object hota hai.
=> when we define a class then at the same time an object of that class is created
that object is called a class object.
=> class object is created in memory when the class is defined.
=> so every class has only one class object.
=> class object is shared by all the objects of the class.
=> class object is used to access the class variables and class methods.
=> class object is used to access the class variables and class methods using the
class name.
class object is collection of class variables and class methods.
```

Example:

```
class Pen:              # class definition + class object creation
    color = 'Blue'
    brand = 'Parker'
    price = 100
# let proof
print(Pen.color)     # Blue
print(Pen.brand)    # Parker
print(Pen.price)    # 100
```

instance object

```
=> jab app class ka object round brackets ke sath create krte ho to wo object
instance object hota hai.
=> when we create an object of a class using parentheses then that object is called
an instance object.
=> instance object is created in memory when the object is created.
=> so every object has its own instance object.
=> instance object is used to access the instance variables and instance methods.
=> instance object firstly is empty but we can add values to it.
```

Example:

```python
class Pen:              # class definition + class object creation
    color = 'Blue'
    brand = 'Parker'
    price = 100

p1 = Pen()              # instance object creation/instantiation
p2 = Pen()              # instance object creation/instantiation

# let proof
print(p1.color)     # Blue
print(p2.color)     # Blue

# we can change the value of instance object
p1.color = 'Red'
print(p1.color)     # Red
print(p2.color)     # Blue

# we can add new properties to instance object
p1.ink = 'Gel'
print(p1.ink)     # Gel
print(p2.ink)     # AttributeError: 'Pen' object has no attribute 'ink'

# we can add new methods to instance object
p1.write = lambda: print('Writing...')
p1.write()     # Writing...
p2.write()     # AttributeError: 'Pen' object has no attribute 'write'

# here is p1 and p2 are reference  variable that are pointing to the object of Pen
class/instance object. or storing the address of the object.
or simply we can say that p1 and p2 are the name of the object of Pen class.
```

----------#

init() method

```
=> it is a special type of function/method that is called initializer or
constructor.as these type of methods are called magic methods/dunder methods.
=> it is one is compulsory to define in a class and you can define more than
one__init__() method in a class but 1 is compulsory.
```

syntax: def **init**(self): # body of **init**() method

```
# Example:
class Pen:
    def __init__(self):         # __init__() method/constructor/initializer
        print('This is __init__() method')    # This is __init__() method
```

Acha Bahi ab deko working :

```
=> 1st class define kia like this
 = > 2nd ma ne class k andar class object variable define kye like
    x=5,y=10
=> 3rd ma ne class k andar __init__() method define kia or us ma self ko
parameter pass kia like this __init__(self)
```

---------- Attributes in class ----------

Note:

we have 4 types of variables in python

1. local variable

2. global variable
3. class object variable(also called static variable) but no static keyword in python like other languages like java,c++ etc. (static keyword is used to define class object variable in other languages
4. instance object variable

Example:

```
# ------------------------------------------------------------------------ #
class pen:
    x = 5
    def f1():
        y = 10
    def __init__(self,a):
        self.a = a
p1 = pen(6)
p2 = pen(8)

# all variables and its type
# 1. x = class object variable because it is defined inside the class but outside
the method
# 2. y = local variable because it is defined inside the method
# 3. a = local variable because it is defined inside the __init__() method
# 4. p1.a = instance object variable because it is defined inside the __init__()
method using self keyword
# 5. p2.a = instance object variable because it is defined inside the __init__()
method using self keyword

# 6. p1 = global variable because it is defined outside the class and inside the
method
# 7. p2 = global variable because it is defined outside the class and inside the
method

# 8. pen = global variable because it is defined outside the class and inside the
method
# 9. f1 = variable to represent function object
# 10. __init__ = variable to represent function object

# totoal 10 variables are there in this example
[pen,x,f1,__init__,p1,p2,y,a,p1.a,p2.a]


# ------------------------------------------------------------------------ #
```

Note:

we have 4 types of methods/function in python

1. instance method
2. static method
3. class method
4. None_Member function(Not a part of class) or global function

```
class pen:
    def __init__(self,a):       # instance method
        self.a = a
    def f2(self):
        pass
    @staticmethod
    def f3():               # static method
        pass
    @classmethod            # class method
    def f4(cls):
        pass
def f5():               # None_Member function(Not a part of class) or global
function
    pass

# totoal 4 methods are there in this example
[__init__,f2,f3,f4,f5]
```

--#

intance method creation and access

Creation of instion ways:

```
class Pen:
    def __init__(self):     # 1 using __init__() method
        pass
    def write(self):        # 2 using instance method
        print('Writing...')
     # using lambda function
    write = lambda self: print('Writing...')  # 3 using lambda function
```

Access of instion ways:

```
``python     p1 = Pen()      # instance object creation     p1.write()      #
Writing...     Pen.write(p1)   # Writing...    ``
```

----------#

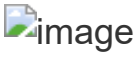
Class object variable creation and access

Creation of class object variable ways:

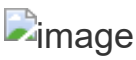
```
class Pen:
   x = 5            # 1 inside the class(static variable)

    def __init__(self):
        pen.y = 10   # 2 using class name  we can access
        # and create class variable using class name

    def write(self):
        Pen.z = 15   # 3 using class name  we can access
        # and create class variable using class name


pen.z = 20          # 4 using class name  we can access
        # and create class variable using class name
#----------------------------------------
Note more important thing :

Example :

class pen:
    def __init__(self,a):       # instance method
        self.a = a
    def f2(self):
        pass
    @staticmethod
    def f3():               # static method
        pass
    @classmethod            # class method
    def f4(cls):
        pass

#---------
```

```python
p1 = pen()
p1.f2()    # f2(p1) => f2(self=p1) we can access instance method using instance object

pen.f3()   # f3() we can access static method using class name


pen.f4()   # f4(pen) beacuse it implicitly pass the class name as a argument we can access class method using class name

#---------
```

```python


class Pen:              # class definition + class object creation
    x = 5               # class object variable
    y = 10              # class object variable

    def __init__(self):       # __init__() method/constructor/initializer
        a=20                  # local variable
        self.z = 15           # instance object variable but how? because we are using self keyword.

p1 = Pen()              # instance object creation/instantiation

# we know that p1 = Pen()  is instance object creation and p1 is reference variable that is pointing to the object of Pen class and it is alway empty firstly



# wehenever we create an object of a class then __init__() method/constructor  is called automatically.like this and in which reference variable is passed as a argument  and this argument recevied p1 to self parameter like blow this

# p1 = Pen() => __init__(p1) method is called automatically like this

# p1 = Pen() => __init__(self=p1) method is called automatically like this

# Now self.z  value store in empty instance object is 15

print(p1.z)     # 15
```



```
iNeuron.ai - Big Data Masters.zip.003Example:2  MORE THAE ONE PARAMETER IN
__init__() METHOD
class Pen:              # class definition + class object creation
    x = 5               # class object variable
    y = 10              # class object variable

    def __init__(self,a):       # __init__() method/constructor/initializer
        self.z = 15           # instance object variable but how? because we are
using self keyword.


p1 = Pen()              # instance object creation/instantiation

# we know that p1 = Pen()  is instance object creation and p1 is reference variable
that is pointing to the object of Pen class and it is alway empty firstly



# wehenever we create an object of a class then __init__() method/constructor  is
called automatically.like this and in which reference variable p1 , a is passed as
a argument  and this argument recevied p1 to self ,valu to a  parameter like blow
this

# p1 = Pen(8) => __init__(p1,8) method is called automatically like this

# p1 = Pen() => __init__(self=p1,a=8) method is called automatically like this

# Now self.z  value store in empty instance object is 15
# Now  8 value store in empty instance object is a

print(p1.z)     # 15
print(p1.a)     # 8

Note:
# wehen we pass more than one parameter in __init__() method then we have to
compulsory pass the value of that parameter/argument when we create an object of
that class like this Pen(8) or Pen(8,9) or Pen(8,9,10) etc.
```



▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬

```
# Example:3  MORE THAE ONE PARAMETER IN __init__() METHOD and i want to store the different value of a in instance object variable with different object creation.


class Pen:              # class definition + class object creation
    x = 5               # class object variable
    y = 10              # class object variable

    def __init__(self,a):       # __init__() method/constructor/initializer
        self.a = a           # instance object variable but how? because we are using self keyword.


p1 = Pen(6)              # instance object creation/instantiation
p2 = Pen(8)              # instance object creation/instantiation

# we know that p1 = Pen() ,p2 = pen() is instance object creation and p1 ,p2 is reference variable that is pointing to the object of Pen class and it is alway empty firstly



# wehenever we create an object of a class then __init__() method/constructor  is called automatically.like this and in which reference variable p1 , a is passed as a argument  and this argument recevied p1 to self ,valu to a  parameter like blow this

# p1 = Pen(8) => __init__(p1,8) method is called automatically like this

# p1 = Pen() => __init__(self=p1,a=8) method is called automatically like this

# Now self.a  value store in empty instance object is 8
# Now  8 value store in empty instance object is a

print(p1.a)     # 8
print(p2.a)     # 6

Note:
# wehen we pass more than one parameter in __init__() method then we have to compulsory pass the value of that parameter/argument when we create an object of that class like this Pen(8) or Pen(8,6) or Pen(8,9,10) etc.

# ----------------------------

# in this example we see that we can store different values of a in instance object variable with different object creation.
main purpose of we => self.a = a  is that we can store the value of a in instance object variable with different object creation.

p1 = Pen(6)      store a = 6
```

```
p2 = Pen(8)      store a = 8
```

----------4-Pillers-Oops----------#

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

---

other concepts

6. Data Hiding (Python does not support Data Hiding concept)
7. Message Passing (Python does not support Message Passing concept)
8. Dynamic Binding (Python does not support Dynamic Binding concept)
9. Operator Overloading (Python does not support Operator Overloading concept)
10. Method Overloading (Python does not support Method Overloading concept)
11. Method Overriding (Python does not support Method Overriding concept)
12. Multiple Inheritance (Python supports Multiple Inheritance concept)
13. Composition (Python supports Composition concept)
14. Aggregation (Python supports Aggregation concept)
15. Association (Python supports Association concept)
16. Dependency (Python supports Dependency concept)

---

1. Encapsulation => Encapsulation is a process of wrapping the data (variables) and code (methods) into a single unit called a class. => Encapsulation is a process of binding the data (variables) and code (methods) into a single unit called an object

=> act of combining properties and methods related to same entity/object.

lets Example

```
class Student:
    # student  properties
    def __init__(self, name, age, rollno,section):
        self.name = name
        self.age = age
        self.rollno = rollno
        self.section = section

    # student methods
    def enrollment:
        pass
    def change_section:
        pass
    def promotion:
        pass

# creating object
s1 = Student('Ali', 20, 101, 'A')
s2 = Student('Ahmed', 21, 102, 'B')

print(s1.name)
print(s2.name)
```

point to understand

```
=> In the above example, we have created a class Student that contains the
properties and methods related to the student.
=> The properties and methods are bunduling  into a single unit called a
class(Student) this concept is called Encapsulation.
=> The properties are name, age, rollno, section, and the methods are enrollment,
change_section, promotion.
```

Note:

```
=> in python properties + methods = is also called Attributes if inside the class
=> like we can say these are Student(class) Attributes[propertis+methods].
```

and in other languages properties is called fields, variables, attributes, characteristics, features, and methods is called functions, behaviors.

---

in python termnology

```
=> properties = variables = fields = attributes = characteristics = features
=> methods = functions = behaviors
```

---

2. Inheritance inheritance mean to extend class