OOPs - Object oriented programming language.

Classes and objects in python

Python is a object oriented programming language. This means that almost all the code is implemented using a special construct called classes. A class is a code template for creating objects.

Class: The class is a user defined data structure that binds the data members and methods into a single unit.

* class is a blueprint or code template for object creation.
* Using a class we can create as many objects as we want.

Object: An object is an instance of a class. It is a collection of attributes (variables) and methods. we use object of a class to perform actions.

ex:- To build a home we need a blue print of the design.

blueprint → class

home → object

* different blue prints give different homes. So different classes create different objects.

* Same blueprint can give houses of different attributes like color, material and such. So single class can create objects having different attributes.

OOPS is a building block for any developer, because if we want to build everything from scratch it takes a very long time.

Programmer without OOPS $\xrightarrow[\text{development}]{\text{app}}$ 1 year

Programmer using all the classes of OOPS $\xrightarrow[\text{development}]{\text{app}}$ 2 months

* Object has two characteristics.
  - states (attributes)
  - behaviours (methods)

  Using its methods we can modify its states.

* Python is completely object oriented language i.e data, functions, lists etc are all objects.

  ex:- 1 is object so Datatype(int) is class.

* All datatypes in python are classes

ex:- 1.2 is object of datatype float class.

ex:- x=1

Print(type(x))

↳ <class 'int'>

i.e when we assign data to a variable python calls integer class and variable x is storing an integer object here. x is also a object here.

ex:- x = [1, 2, 3]

print(type(x))

↳ <class 'list'>

ex:- x = f1 → function assigned to variable, both function and variable are objects

x()

↙

when we call the function it creates an object.

Python (Final Definition): Python is a case-sensitive high level object oriented programming language.

Creator of python: Guido Van Rossum.

Create a class:

Class is defined using the 'class' keyword.

Syntax:

```
Class ClassName:        <- no brackets as it is not a function
    """This is a docstring"""
    <statement 1>
    <statement 2>
    .
    .
    .
    <statement n>
```

* ClassName: is the name of the class. It follows Pascal naming convention.
* Docstring: It is the first string inside the class and has brief description of the class. Although not mandatory this is highly recommended.
* Statements: Attributes and methods. All classes can only contain Attributes and methods.

Create object of a class:

* An object is essential to work with the attributes of the class. The object is created using the class name.
* When we create an object of a class. It is called Instantiation. The object is called instance of a class.

* A constructor is a special method used to create and initialize an object of a class. This method is defined in the class.

In python object creation is divided in to two parts in object creation and object initialization.

* Internally the __new__ is the method that creates the object.
* And using __init__() method we can implement constructor to initialize the object.

Syntax

<Object-name> = <ClassName>(<arguments>)

Simple ex:-

```
class First:        → create a class
    pass

ob = First()        calling the class
                    → creates an object
```

↳ object that belongs to First class

```
print(type(ob))
```

↳ <class '__main__.First'>

Simple examples of attributes and methods:

If we consider mobile:

attributes - 2 cameras
- charger port
- screen

methods - can take pictures
- gets charged
- makes calls.

If we consider car:

attributes - 4 tyres
- Airbags
- doors
- dash board
- steering wheel

methods - can drive
- can play music.

If we consider Human beings:

God has created all Human beings with a blue print. so humans are from a single class so they share simillarities and also differences like body color, skin color, height, name, age etc.

Similar attributes	
2- eyes	2 - legs
1- nose	2- hands
2- ears	

So attributes are variables to which we are giving some values.

ex:- eyes = 2

ears = 2

methods are actions performed by the object.

ex:- walk

talk

eat

⋮

** functions when called inside the class are called methods.

** methods always take a first parameter 'self'.

In Jupyter Notebook

If we type some variable and use dot operator. After '.' if we double press tab we can see all the methods we can apply ex:- l = [1, 2, 3, 4]

l.

↘ double click tab.

* All the attributes and methods inside the class can be accessed by the object created by the function.

** __init__ this is a magic constructor method used to initialize something.

This is a special constructor which will be automatically called when creating the object.

This is not called by the programmer.

default of this is:

```
def __init__(self):
    pass
```

this initilizes the object when class is called and we can edit it to add attributes.

ex:-

```
class HumanBeing:
    def __init__(self, color)
        self.color = color
    def walk(self):
        print("I can walk")
    def talk(self):
        print("I am {}".format(self.color))
```

instance ←

```
P1 = HumanBeing("black")
P2 = HumanBeing("white")
P3 = HumanBeing("brown")
```

↓ objects

} from same class but different objects

↓ instance objects

↓ instance attributes

P1.talk()
↳ I am black

P2.talk()
↳ I am white

P3.talk()
↳ I am brown

P2.walk()
↳ I can walk

Class attributes are classified to two types.
- Instance variable (attribute)
- Class variable (attribute)

Instance Variable - They are from same class but are different for each object.
They are unique to that particular object.

Class Variable . These are the properties that will be same for all objects coming from a class.

Ex:- eyes = 2
nose = 1
ears = 2
hands = 2
:

```
class HumanBeing:
    # class attributes
    eyes = 2
    legs = 2
    hands = 2
    # instance attributes
    def __init__(self, color, hair-color)
        self.color = color
        self.hair-color = hair-color
    def walk(self):
        print("I can walk")
    def talk(self):
        print("my body color is {} and my hair color
              is {}".format(self.color, self.hair-color))

P1 = HumanBeing("black", "white")
P1.talk()
```

Instance attribute ←

↳ my body color is black and my hair color is white.

```
P1.eyes
```

↳ 2

* we can also have static variable. Only one copy of static variable is created and shared between all the objects of the class.

Class Attributes

Instance Variables

1) Bound to object
2) Declared inside the __init__() method
3) Not shared by objects. Unique for every Object.

Class variables

1) Bound to the class
2) Declared inside class but outside any method.
3) shared by all objects of a class.

Self: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

It does not have to be named 'self', we can call it whatever we like but it has to be first parameter of any method in the class and has to be consistent.

when we are calling

```
class C1:
    def walk(self):
        ------

O = C1()
```

↳ internally object is created and stored in memory. For this reference is self.

```
O.walk
```

↳ here 'self' is redirecting 'O' using the reference to access walk method.

* self is the reference to the object to access the attributes and methods inside the class.

Accessing properties and assigning values:

* An instance attribute can be accessed or modified by using the dot notation (dot operator)

  instance-name.attribute-name

  ex:- p1.color

* A class variable is accessed or modified using the class name.

  ex:- HumanBeing.eyes

```
class HumanBeing:
    # class attributes
    nose = 1
    eyes = 2
    legs = 2
    hands = 2
    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def talk(self):
        print("{}_{}".format(self.name, self.age))
```

} class attributes

} instance attributes

← initialize

Creating objects

per1 = HumanBeing ("ajay", 23)

per2 = HumanBeing ("jay", 41)

accessing attributes

per1.eyes

↳ 2

per1.talk

↳ ajay 23

if we want to change instance attributes

per1.name

↳ ajay

per1.name = "Rohan"

per1.name

↳ Rohan

per2.name

↳ jay

(Per2 name will not change only per 1 name changes)

if we want to change class attribute using Object

per1.eyes = 5

per1.eyes

↳ 5

per2.eyes

↳ 2

(this is just a temporary change of attribute for that Object. Here class attribute eyes acts like instance attribute)

if we want to change class attribute we need to use class name

HumanBeing.eyes = 10

per1.eyes
↳ 10

per2.eyes
↳ 10

* Python is dynamically typed language so after creating the class and objects also we can dynamically add attributes.
* when we are dynamically adding it will first check if attribute is already present or not.

if we want to add new instance attribute

per1.height = 170.0
→ newly added

per1.height
↳ 170.0

per2.height
↳ X (error, instance attribute height only added to object 'per1')

if we want to add new class attribute

HumanBeing.ears = 2

per1.ears
↳ 2

per2.ears
↳ 2

* attributes are properties.
* methods are functionalities.
* self.attribute-name is the instance variable or instance attribute

Methods

- Instance methods
- Class methods

In python object-oriented programming, when we design a class, we use the instance methods and class methods.

Instance methods: used to access or modify the object state. If we use instance variables inside a method, such methods are called instance methods. It must have 'self' parameter.

* The instance method performs a set of actions on the data/value provided by the instance variables.
* A instance method is bound to the object of the class.
* It can access or modify the object state by changing the value of a instance variables.
* When we create a class in python, instance methods are used regularly.
* To work with an instance method, we use 'self' keyword. The 'self' refers to current object.

* Any method we create in class will automatically be created as an instance method unless we explicitly tell python otherwise.

ex:-

```
class HumanBeing:
    # instance attributes
    def __init__(self, name, age):   ← initialize
        self.name = name
        self.age = age                } instance attributes

    def talk(self):   ← instance method
        print("{}_{}".format(self.name, self.age))

# Creating objects
per1 = HumanBeing("ajay", 23)
per2 = HumanBeing("jay", 41)
# calling instance method
per1.talk()
      ↑ call instance method.
```

* Instance methods that are called using the object. They cannot be called using the className.

Class methods :

Class methods are methods that are called on the class itself, not on a specific object instance. Therefore it belongs to a class level, and all the class instances share a class method.

* Any method created in a class will automatically be created as an instance method. If we want these instance methods to act as class methods we must explicitly use decorator called @classmethod which modifies the function.

ex:-

```
class HumanBeing:
    # class attributes
    hands = 2
    eyes = 2
    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def talk(self):
        print("{}_{}".format(self.name, self.age))
    # defining class methods
```

class attributes

instance attributes

← initialize

class method

```
@classmethod
    def see(cls):
        print("I can see")

@classmethod
    def eat(cls):
        Print("I can eat with my hands")
```

* Class method will have a different reference called 'cls'.
* Every time we want to write a class method we should start with decorator. @classmethod.
* we mostly use class methods when all the objects have same actions.
* The class method can be accessed by both class and objects.

ex:-

```
HumanBeing.eat()
    ↳ I can eat with my hands
HumanBeing.see()
    ↳ I can see
Per1.see()
    ↳ I can see.
```

Inheritance in Python:

In python we have a concept called 'DRY' which is 'Do not Repeat Yourself.'

so for example:

```
class HumanBeing:
    def breathe(self):
        - - - - -
    def eat(self):
        - - - - -
    def see(self):
        - - - - -
```

```
class Animals:
    def breathe(self):
        - - - - -
    def eat(self):
        - - - - -
    def see(self):
        - - - - -
```

As we see lot of functionalities and actions are simillar between class HumanBeing and class Animals. so instead of repeating the code we write the common code in a different class and other classes can take it from there and use it.

* The process of inheriting the properties of the parent class in to a child class is called <u>Inheritance</u>.

* The existing class is called a base class or Parent class and the new class is called Subclass or Childclass or derived class.

ex:-

```
class LivingThing:
    def breathe(self):
        - - - - -
    def eat(self):
        - - - - - -
    def see(self):
        - - - - - -
```

Parent class or Base class

```
Class HumanBeing(LivingThing)
    pass

Class Animals(LivingThing)
    pass
```

child class or Sub class or derived class

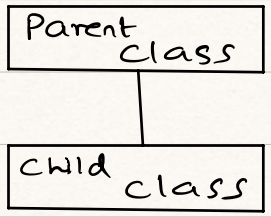
* Inheritance is used of reusability of the code.
* Inheritance child class acquires all the attributes and methods from the parent class.

Different types of Inheritance are :-

- Single Inheritance
- multiple Inheritance
- multi level Inheritance

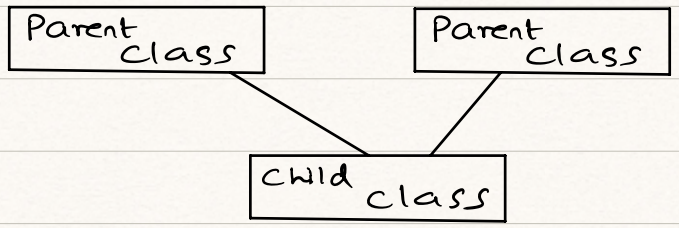
Single Inheritance :

A child class inherits from a single parent class. Here is one child class and one parent class.



multiple Inheritance :

In multiple Inheritance one child can inherit from multiple parent class. so here it is one child and multiple parent classes.



ex:-

```
Class PA1:
    x=1
    def f1(self):
        print('hi')
```

```
Class PA2:
    y=2
    def f1(self):
        print('bye')
Class ch1(PA1, PA2):
    pass
```

this defines the precedence of PA1 over PA2

```
Ob1 = ch1()
Ob1.x
    → 1
Ob1.y
    → 2
Ob1.f1()
    → hi
```

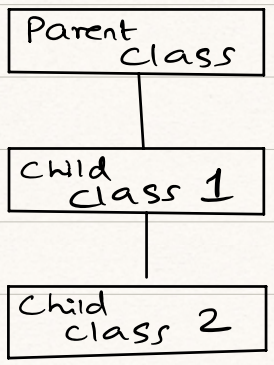
→ as PA1 function f1() has precedence over PA2 function f1() 'hi' is printed instead of 'bye'.

* If some of the parents have same attributes or methods preference will be given based on how the parent classes are passed to child class.

child-class(first preference, next preference, ...)

Multi-level Inheritance:

In multilevel inheritance a class inherits from a child class or derived class. Suppose three classes A, B, C. A is super class, B is child of a class A, C is the child of class B. In otherwords we can say a chain of classes is called multilevel inheritance.



* here child class 2 will acquire attributes and functions of both parent class and child class 1.
* child class 1 will only acquire attributes and methods of parent class.
* multi-level inheritance is not used extensively in python big codes.

Method overriding and method overloading:

Method overriding:

When a method in the child class (sub class) has the same name, same parameters and same return type as a method in parent class (super class), then the method in the child class is said to override the method in parent class.

The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed. But if an object of the child class is used to invoke the method, then the version in the child class will be executed.

ex:-

```
class P:
    def __init__(self):
        print('hi')

class C(P):
    def __init__(self):
        print('bye')
```

```
o = C()
  ↳ bye
```

(Child class method overriding the parent class method)

```
o1 = P()
  ↳ hi
```

(Parent class method overriding child class method)

* Here __init__() was explicitly mentioned so overriding concept was applied.

```
class P:
    def __init__(self):
        print('hi')
    def f1(self):
        print("pclass")

class C(P):
    def f1(self):
        print("cclass")
```

```
o = C()
  ↳ bye
```

(only parent class has __init__ defined)

```
o.f1()
  ↳ c class
```

(child class method overriding parent class method)

* If instead of child class method overriding parent class we want child class object to use parent class method we can use an explicit function called super() function.

```
Class P:
    def __init__(self):
        print('hi')
    def f1(self):
        print("pclass")

Class C(P):
    def __init__(self):
        super().__init__()
        print('bye')

O = C()
```

↳ hi
bye

(super() function accesses parent class methods)

```
Class P:
    def __init__(self):
        print('hi')
```

```
    def f1(self):
        print ("pclass")

class C(P):
    def __init__(self):
        print ('bye')
        super().__init__()
```

```
o = C()
```
↳ bye
hi

(order depends on what statement is executed first)

```
class P:
    def __init__(self):
        print ('hi')
    def f1(self):
        print ("pclass")

class C(P):
    def f1(self):
        super().f1()
        print ("cclass")
```

```
o = C()
```
↳ hi

o.f1()

↳ pclass

cclass

Super( ) function syntax

Super( ). method-name( )

↳ this is the method of super class we want to use.

ex:- Super( ). f1( )

Method overloading :

Two or more methods in the same class have the same name but different number of parameters or different types of parameters, or both. These methods are called overloaded methods and this is called method overloading.

* In python method overloading is not supported. Instead the latest method gets executed.
* Python only cares about function name & not the no. of parameters unlike Java & C++.

```
class P:
    def f1(self, a):
        print(a)
    def f1(self, a, b):
        print (a, b)

q = P()
q.f1(1)
```

↳ error (expecting 2 parameters as 2nd method is executed)

```
q.f1(1,2)
```

↳ 1 2

```
class p:
    def f1(self, a):
        print(a)
    def f1(self, a, b):
        print (a, b)
    def f1(self, a, b, c):
        print (a, b, c)

q = p()
q.f1(1, 2)
```

↳ error (expecting 3 parameters as 3rd method is executed)

```
q.f1(1, 2, 3)
```

↳ 1 2 3

* If we want method overloading to work in Python we can take help of a library to execute method overloading.

1st – pip install multipledispatch

(Install multiple dispatch library. If it is already installed or once it is installed import dispatch module from it).

2nd – from multipledispatch import dispatch

Here by using multiple dispatch decorator (@dispatch) we can make method overloading to work.

```
class p:
    @dispatch(int):
    def f1(self,a):
        print(a)
    @dispatch(int,int):
    def f1(self,a,b):
        print(a,b)
    @dispatch(int,int,int)
    def f1(self,a,b,c):
        print(a,b,c)
```

* Now python focuses on 3 things.
  1) function name
  2) how many parameters are being passed.
  3) data type of the arguments given. If datatype does not match we will get error.

```
o=p()
o.f1(1)
  ↳ 1
o.f1(1,2)
  ↳ 1 2
```

only working because of @dispatch decorator function

o.f1(1, 2, 3)
↳ 1 2 3

o.f1(1, 2, 3.2)
↳ error (3.2 is float not int)

** In python all the classes are originally the children of 'object' class.

* so 'object' class is the super class of all the classes in python.

* 'object' class has all the magical functions.

* In Jupyter Notebook we are not able to see the magic functions. It is visible in visual Studio.

ex:- class P:
pass

q = P()

q. (double tap shows all magical functions)

* issubclass(class1, class2)
↳ True if class1 is child of class2.

ex:- issubclass(P, object)
↳ True

* So all the classes in python already a child of single inheritance.

* isinstance (data, datatype)

  ↳ True if data is of that datatype.

  ex:- isinstance (1, int)

  ↳ True

  isinstance (1.2, int)

  ↳ False

* Polymorphism
* Abstraction
* encapsulation

} Advanced