OOPs concept

Object

- An object is a particular instance of a class with unique characteristics and functions.
- An object is an entity that has attributes and behaviour.

- Object =Car
  - Attribute=wheel, seat, engine
  - Behavior=driving, accelerating

Syntax:

```
# Declare an object of a class
object_name = Class_Name(arguments)
```

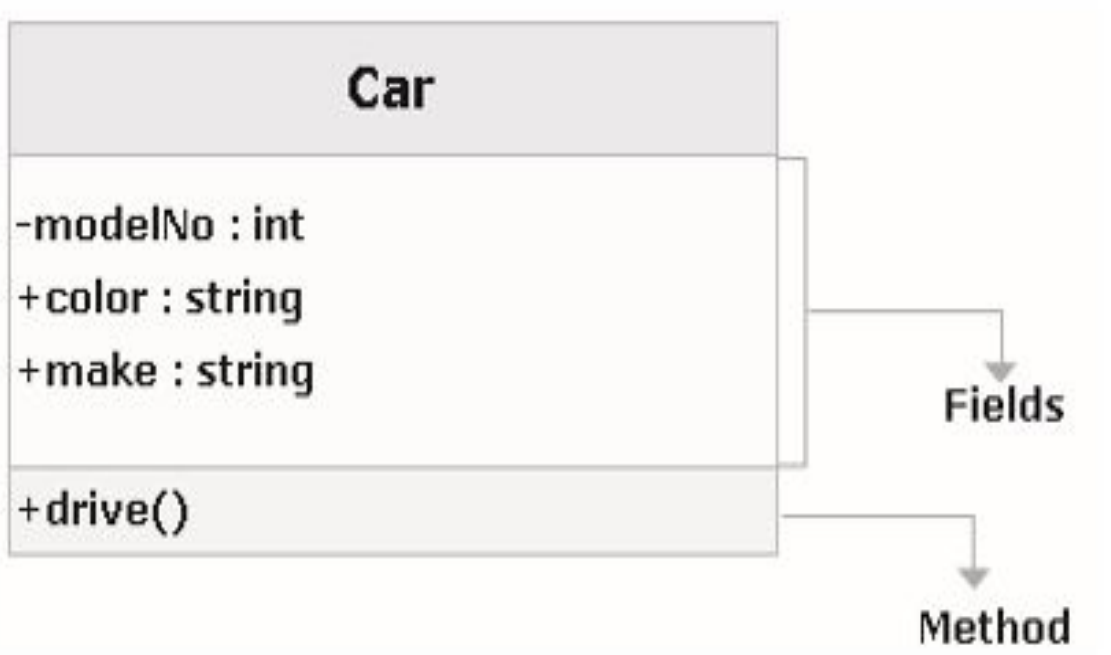
Class

- a class is a user-defined data type that contains both the data itself and the methods that may be used to manipulate it.
- A class is a blueprint for an objects.
- A class as well as its instance are namespaces.

Creating Classes in Python

Syntax :

```
class ClassName:
    #statement_suite
```




```
class simplest:
    pass
s=simplest()    #create an instance of simplest
```

```
class car:
    Modelno=2000
    Color=red
    Make=2/6/2020
    def drive():
        print("driving")

obj=Car()
obj.drive()
Print(obj.ModelNo)
```

Methods

- As we discussed above, an object has attributes and behaviors. These behaviors are called methods in programming.

Example:

we have two objects Ram and Steve that belong to the class Human

Object attributes: name, height, weight

Object behavior: eating()

- **The self-parameter**

  The self-parameter refers to the current instance of the class and accesses the class variables. We can use anything instead of self, but it must be the first parameter of any function which belongs to the class.

Constructor

A constructor is a special kind of function which is used for initializing the instance variables during object creation.

Syntax ::

A constructor always has a name init and the name init is prefixed and suffixed with a double underscore(__). We declare a constructor using def keyword,

just like methods.

```
def __init__(self):
    # body of the constructor
```

Types of constructors in Python

We have two types of constructors in Python.

1. default constructor – this is the one, which we have seen in the above example. This constructor doesn't accept any arguments.
2. parameterized constructor – constructor with parameters is known as parameterized constructor.

default constructor

An object cannot be created if we don’t have a constructor in our program. This is why when we do not declare a constructor in our program, python does it for us.

Example: When we do not declare a constructor

In this example, we do not have a constructor but still we are able to create an object for the class. This is because there is a default constructor implicitly injected by python during program compilation,:

```
def __init__(self):
    # no body, does nothing
```

Parameterized constructor example

- When we declare a constructor in such a way that it accepts the arguments during object creation then such type of constructors are known as Parameterized constructors.

```
class DemoClass:
    num = 101

    # parameterized constructor
    def __init__(self, data):
        self.num = data


    def read_number(self):
        print(self.num)

obj = DemoClass(55)

# calling the instance method using the object obj
obj.read_number()

# creating another object of the class
obj2 = DemoClass(66)

# calling the instance method using the object obj
obj2.read_number()
```

SN Function Description

1 getattr(obj,name,default): It is used to access the attribute of the object.

2 setattr(obj, name,value): It is used to set a particular value to the specific attribute of an object.

3 delattr(obj, name): It is used to delete a specific attribute.

4 hasattr(obj, name): It returns true if the object contains some specific attribute.

Attribute	Description
__dict__	This is a dictionary holding the class namespace.
__doc__	This gives us the class documentation if documentation is present. None otherwise.
__name__	This gives us the class name.
__module__	This gives us the name of the module in which the class is defined. In an interactive mode it will give us __main__.
__bases__	A possibly empty tuple containing the base classes in the order of their occurrence.

Python Destructor

Destructors are called when an object gets destroyed. It’s the polar opposite of the constructor, which gets called on creation.

Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Python Garbage Collection

- Python deletes unneeded objects automatically to free the memory space.
- The process by which Python periodically reclaims blocks of memory that no longer are in use is termed as Garbage Collection.
- Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero.
- An object's reference count changes as the number of aliases that point to it changes.

Python Inheritance

- Inheritance is an important aspect of the object-oriented paradigm. Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch.

Syntax

class derived-class(base class):

    <class-suite>

Syntax ::

class derive-class(<base class 1>, <base class 2>, ... <base class n>):

    <class - suite>

Inheritance Type

- **Single inheritance is** only one super class inherit into the one base class
- **Multiple Inheritance** is more than one super class inherit into the one derived class.
- **Multi-Level** Multi-level inheritance is archived when a derived class inherits another derived class.
- **Hierarchical inheritance** More than one derived classes are created from a single base.
- **Hybrid inheritance**: This form combines more than one form of inheritance. Basically, it is a blend of more than one type of inheritance.

Python Multi-Level inheritance

- Multi-Level inheritance is possible in python like other object-oriented languages.

Syntax

```
class class1:
    <class-suite>
class class2(class1):
    <class suite>
class class3(class2):
    <class suite>
```

Python Multiple inheritance

Python provides us the flexibility to inherit multiple base classes in the child class.

Syntax

```
class Base1:
   <class-suite>
 class Base2:
   <class-suite>


class BaseN:
   <class-suite>


class Derived(Base1, Base2, ..... BaseN):
   <class-suite>
```

Python hierchical inheritance
The issubclass(sub,sup) method

The issubclass(sub, sup) method is used to check the relationships between the specified classes. It returns true if the first class is the subclass of the second class, and false otherwise.

The isinstance (obj, class) method

The isinstance() method is used to check the relationship between the objects and classes.

Python isinstance()

Python isinstance() function syntax is:

isinstance(object, classinfo)

This function returns True if the object is instance of classinfo argument or instance of classinfo subclass.

If the object is not an instance of classinfo or its subclass, then the function returns False.

classinfo argument can be a tuple of types. In that case, isinstance() will return True if the object is an instance of any of the types.

If classinfo is not a type or tuple of types, a TypeError exception is raised.

Method Overriding in Python

- Method overriding is a concept of object oriented programming that allows us to change the implementation of a function in the child class that is defined in the parent class.
- It is the ability of a child class to change the implementation of any method which is already provided by one of its parent class(ancestors).

Method Overriding

- We can provide some specific implementation of the parent class method in our child class.

Example

```
class Animal:
    def speak(self):
        print("speaking")
class Dog(Animal):
    def speak(self):
        print("Barking")
d = Dog()
d.speak()
```

1. Inheritance should be there.
2. Function overriding cannot be done within a class.
3. We need to derive a child class from a parent class.
4. The function that is redefined in the child class should have the same signature as in the parent class i.e. same number of parameters.

operator overloading

Python operators work for built-in classes.

But same operator behaves differently with different types.

For example, the + operator will, perform arithmetic addition on two numbers, merge two lists and concatenate two strings.

Operator Overloading Special Functions in Python		
Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)

Comparision Operator Overloading in Python

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

Difference between public, private and protected

Mode	Description
public	A public member is accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function. By default all the members of a class would be public
private	A private member variable or function cannot be accessed, or even viewed from outside the class. Only the class members can access private members. Practically, we make data private and related functions public so that they can be called from outside of the class
protected	A protected member is very similar to a private member but it provided one additional benefit that they can be accessed in sub classes which are called derived/child classes.