Python Object Oriented Programming

**Introduction to OOPs in Python**

Python is a multi-paradigm programming language. Meaning, it supports different programming approach.

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

* attributes
* behavior

Let's take an example:

Parrot is an object,

* name, age, color are attributes
* singing, dancing are behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

|  |  |
| --- | --- |
| Inheritance | A process of using details from a new class without modifying existing class. |
| Encapsulation | Hiding the private details of a class from other objects. |
| Polymorphism | A concept of using common operation in different ways for different data input. |

**Class**

A class is a blueprint for the object.

We can think of class as an sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, parrot is an object.

The example for class of parrot can be :

class Parrot:

pass

Here, we use **class** keyword to define an empty class **Parrot**. From class, we construct instances. An instance is a specific object created from a particular class.

**Object**

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

**obj = Parrot()**

Here, obj is object of class Parrot.

Suppose we have details of parrot. Now, we are going to show how to build the class and objects of parrot.

**Example 1: Creating Class and Object in Python**

class Parrot:

# class attribute

species = "bird"

# instance attribute

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

# instantiate the Parrot class

blu = Parrot("Blu", 10)

woo = Parrot("Woo", 15)

# access the class attributes

print("Blu is a {}".format(blu.\_\_class\_\_.species))

print("Woo is also a {}".format(woo.\_\_class\_\_.species))

# access the instance attributes

print("{} is {} years old".format( blu.name, blu.age))

print("{} is {} years old".format( woo.name, woo.age))

Run

When we run the program, the output will be:

Blu is a bird

Woo is also a bird

Blu is 10 years old

Woo is 15 years old

In the above program, we create a class with name Parrot. Then, we define attributes. The attributes are a characteristic of an object.

Then, we create instances of the Parrot class. Here, blu and woo are references (value) to our new objects.

Then, we access the class attribute using \_\_class \_\_.species. Class attributes are same for all instances of a class. Similarly, we access the instance attributes using blu.name and blu.age. However, instance attributes are different for every instance of a class.

To learn more about classes and objects, go to [Python Classes and Objects](https://www.programiz.com/python-programming/class)

**Methods**

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

**Example 2 : Creating Methods in Python**

class Parrot:

# instance attributes

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

# instance method

def sing(self, song):

return "{} sings {}".format(self.name, song)

def dance(self):

return "{} is now dancing".format(self.name)

# instantiate the object

blu = Parrot("Blu", 10)

# call our instance methods

print(blu.sing("'Happy'"))

print(blu.dance())

Run

When we run program, the output will be:

Blu sings 'Happy'

Blu is now dancing

In the above program, we define two methods i.e sing() and dance(). These are called instance method because they are called on an instance object i.e blu.

**Inheritance**

Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

**Example 3: Use of Inheritance in Python**

# parent class

class Bird:

def \_\_init\_\_(self):

print("Bird is ready")

def whoisThis(self):

print("Bird")

def swim(self):

print("Swim faster")

# child class

class Penguin(Bird):

def \_\_init\_\_(self):

# call super() function

super().\_\_init\_\_()

print("Penguin is ready")

def whoisThis(self):

print("Penguin")

def run(self):

print("Run faster")

peggy = Penguin()

peggy.whoisThis()

peggy.swim()

peggy.run()

Run

When we run this program, the output will be:

Bird is ready

Penguin is ready

Penguin

Swim faster

Run faster

In the above program, we created two classes i.e. Bird (parent class) and Penguin (child class). The child class inherits the functions of parent class. We can see this from swim()method. Again, the child class modified the behavior of parent class. We can see this from whoisThis() method. Furthermore, we extend the functions of parent class, by creating a new run() method.

Additionally, we use super() function before \_\_init\_\_() method. This is because we want to pull the content of \_\_init\_\_() method from the parent class into the child class.

**Encapsulation**

Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single “ \_ “ or double “ \_\_“.

class Computer:

def \_\_init\_\_(self):

self.\_\_maxprice = 900

def sell(self):

print("Selling Price: {}".format(self.\_\_maxprice))

def setMaxPrice(self, price):

self.\_\_maxprice = price

c = Computer()

c.sell()

# change the price

c.\_\_maxprice = 1000

c.sell()

# using setter function

c.setMaxPrice(1000)

c.sell()

Run

When we run this program, the output will be:

Selling Price: 900

Selling Price: 900

Selling Price: 1000

In the above program, we defined a class Computer. We use \_\_init\_\_() method to store the maximum selling price of computer. We tried to modify the price. However, we can’t change it because Python treats the \_\_maxprice as private attributes. To change the value, we used a setter function i.e setMaxPrice() which takes price as parameter.

**Polymorphism**

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.

class Parrot:

def fly(self):

print("Parrot can fly")

def swim(self):

print("Parrot can't swim")

class Penguin:

def fly(self):

print("Penguin can't fly")

def swim(self):

print("Penguin can swim")

# common interface

def flying\_test(bird):

bird.fly()

#instantiate objects

blu = Parrot()

peggy = Penguin()

# passing the object

flying\_test(blu)

flying\_test(peggy)

Run

[Powered by DataCamp](https://www.datacamp.com/)

When we run above program, the output will be:

Parrot can fly

Penguin can't fly

In the above program, we defined two classes Parrot and Penguin. Each of them have common method fly() method. However, their functions are different. To allow polymorphism, we created common interface i.e flying\_test() function that can take any object. Then, we passed the objects blu and peggy in the flying\_test() function, it ran effectively.

**OOPs:**

---------

classes

A class is a collection of data and functions

data : usually private

functions : usually public

objects

An object is an instance of a class

**Creating a class:**

**Syntax:**

class className:

variables : instance variables, static and local variables

methods : instance methods, static methods, class methods

**Ex:**

class Student:

def setName(name):

...

def getName():

rerturn name

**JAVA**

--------

class Student

{

String name;

public void setname(String s)

{

name=n;

}

}

//Creating object

Student s =new Student();

s.setname("Rajendra");

or

class Student

{

String name;

public Student(String s)

{

name=n;

}

}

Student s =new Student("Rajendra");

Physical existence of a class is nothing but object. We can create any number of objects for a class.

Syntax

referencevaraible=classname()

s=Student()

**What is Reference Variable ?**

-------------------------------------------

The variable which can be used to refer object is called reference variable.

By using reference variable, we can access properties and methods of object.

**Self variable:**

--------------------

1. self is the default variable which is always pointing to current object (like this keyword in Java)

2. By using self we can access instance variables and instance methods of object

**Note:**

1. self should be first parameter inside constructor

def \_\_init\_\_(self):

2. self should be first parameter inside instance methods

def talk(self):

**Constructor**

-------------------

1. Constructor is a special method in Python

2. The name of the constructor should be \_\_init\_\_(self)

3. Whenever an object gets build up, a constructor get called automatically

4. The main purpose of constructor is to declare and initialize instance variables

5. Per object constructor will be called only once

6. Constructor can take atleast one argument(self)

**Example:1**

class Student:

def \_\_init\_\_(self):

self.name='Rajendra'

self.age=42

self.marks=78

def talk(self):

print(self.name)

print(self.age)

print(self.marks)

print()

s1=Student()

s1.talk()

s2=Student()

s2.talk()

Differnt objects, Different data

Example:

class Student:

def \_\_init\_\_(self,name,age,marks): # Parameterized constructor(3-arg)

self.name=name

self.age=age

self.marks=marks

def talk(self):

print(self.name)

print(self.age)

print(self.marks)

print()

s1=Student('Rajendra',42,99)

s1.talk()

s2=Student('Sheetal',24,78)

s2.talk()

**Introduction**

-----------------

- General purpose high level language

- Combines the features of C and Java

- Devloped by Van Rossum >>1989 >>1991

- Monty Python Flying Circus

**Features of Python**

-----------------

1. Simple

2. Easy to learn

3. Open source

4. High level language

5. Dynamically typed

**C**

main()

{

int a,b,sum;

a=10;

b=20;

sum=a+b;

printf("%d",sum);

}

**Python**

a=10

b=20

sum=a+b

print(sum)

6. Platform independent

7. Portable

8. Procedure and Object Oriented

9. Extensible

10. Embeddable

11. Huge Library

12. Database connectivity

13. Batteries included

- numpy (Numerical Python)

- pandas (Data Analysis)

- matplotlib(Graphics)

**Where we can use Python**

-----------------------------------

1. Desktop based applications

2. Web Applications(DJango, Flask )

3. Network applications

4. Data Analysis

5. AI

6. Machine Learning

7. IOT operations...

**Which companies**

------------------------

1. Google

2. DropBox

3. Youtube

4. NASA

**Frozen Binaries**

-------------------

Converting the Python programs into true executables is called frozen binaries

For creating frozen binaries, we need third pary software

py2exe : Windows OS

pyinstaller : UNIX/Linux

**Garbage Collector**

-----------------------

Module : gc

Going Forward

---------------------

1. Data Types in Python

2. Operators in Python

3. Input and Output Statements

4. Flow Control

5. Arrays

numpy

6. Strings

7. Functions

8. Lists, Tuples, Dictionaries

9. OOPs

classes and objects

Inheritance

Polymorphism

Abstract methods, Abstract classes

Interfaces

Exceptions

10. Files

11. Regular Expressions

12. Threads

13. Graphical USer Interface

14. Database Connectivity