



Python OOPs Concepts

In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

OOPs Concepts in Python

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction



Python Class

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, and age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.:
Myclass.Myattribute

Class Definition Syntax:

```
class ClassName:  
    # Statement-1  
    .  
    .  
    .  
    # Statement-N
```

Creating an Empty Class in Python

In the above example, we have created a class named Dog using the class keyword.

Python

```
# Python3 program to  
# demonstrate defining  
# a class  
  
class Dog:  
    pass
```

Python Objects

The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 12 is an object, the string “Hello, world” is an object, a list is an object that can hold other objects, and so on. You’ve been using objects all along and may not even realize it.

An object consists of:

- **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
- **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

To understand the state, behavior, and identity let us take the example of the class dog (explained above).

- The identity can be considered as the name of the dog.
- State or Attributes can be considered as the breed, age, or color of the dog.
- The behavior can be considered as to whether the dog is eating or sleeping.

Creating an Object

This will create an object named obj of the class Dog defined above. Before diving deep into objects and classes let us understand some basic keywords that will we used while working with objects and classes.

Python3

```
obj = Dog()
```

The Python self

1. Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it
2. If we have a method that takes no arguments, then we still have to have one argument.
3. This is similar to this pointer in C++ and this reference in Java.

When we call a method of this object as `myobject.method(arg1, arg2)`, this is automatically converted by Python into `MyClass.method(myobject, arg1, arg2)` – this is all the special `self` is about.

Note: For more information, refer to [self in the Python class](#)

The Python __init__ Method

The [__init__ method](#) is similar to constructors in C++ and Java. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object. Now let us define a class and create some objects using the `self` and `__init__` method.

Creating a class and object with class and instance attributes

Python3

```
class Dog:

    # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name

# Driver code
```

```
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")
```

```
# Accessing class attributes
print("Rodger is a {}".format(Rodger.__class__.attr1))
print("Tommy is also a {}".format(Tommy.__class__.attr1))

# Accessing instance attributes
print("My name is {}".format(Rodger.name))
print("My name is {}".format(Tommy.name))
```

Output

```
Rodger is a mammal
Tommy is also a mammal
My name is Rodger
My name is Tommy
```

Creating Classes and objects with methods

Here, The Dog class is defined with two attributes:

- attr1 is a class attribute set to the value “mammal”. Class attributes are shared by all instances of the class.
- __init__ is a special method (constructor) that initializes an instance of the Dog class. It takes two parameters: self (referring to the instance being created) and name (representing the name of the dog). The name parameter is used to assign a name attribute to each instance of Dog.

The speak method is defined within the Dog class. This method prints a string that includes the name of the dog instance.

The driver code starts by creating two instances of the Dog class: Rodger and Tommy. The __init__ method is called for each instance to initialize their name attributes with the provided names. The speak method is called in both instances (Rodger.speak() and Tommy.speak()), causing each dog to print a statement with its name.

Python3

```
class Dog:

    # class attribute
    attr1 = "mammal"

    # Instance attribute
```

```

def __init__(self, name):
    self.name = name

def speak(self):
    print("My name is {}".format(self.name))

```

```

# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class methods
Rodger.speak()
Tommy.speak()

```

Output

```

My name is Rodger
My name is Tommy

```

Note: For more information, refer to [Python Classes and Objects](#)

Python Inheritance

Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class. The benefits of inheritance are:

- It represents real-world relationships well.
- It provides the reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Types of Inheritance

- **Single Inheritance:** Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.
- **Multilevel Inheritance:** Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.
- **Hierarchical Inheritance:** Hierarchical-level inheritance enables more than one derived class to inherit properties from a parent class.
- **Multiple Inheritance:** Multiple-level inheritance enables one derived class to inherit properties from more than one base class.

Inheritance in Python

In the above article, we have created two classes i.e. Person (parent class) and Employee (Child Class). The Employee class inherits from the Person class. We can use the methods of the person class through the employee class as seen in the display function in the above code. A child class can also modify the behavior of the parent class as seen through the details() method.

Python3

```
# Python code to demonstrate how parent constructors
# are called.

# parent class
class Person(object):

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))

# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))

# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")

# calling a function of the class Person using
# its instance
a.display()
a.details()
```

Output

Rahul
886012
My name is Rahul
IdNumber: 886012
Post: Intern

Note: For more information, refer to our [Inheritance in Python](#) tutorial.

Python Polymorphism

Polymorphism simply means having many forms. For example, we need to determine if the given species of birds fly or not, using polymorphism we can do this using a single function.

Polymorphism in Python

This code demonstrates the concept of inheritance and method overriding in Python classes. It shows how subclasses can override methods defined in their parent class to provide specific behavior while still inheriting other methods from the parent class.

Python3

```
class Bird:

    def intro(self):
        print("There are many types of birds.")

    def flight(self):
        print("Most of the birds can fly but some cannot.")

class sparrow(Bird):

    def flight(self):
        print("Sparrows can fly.")

class ostrich(Bird):

    def flight(self):
        print("Ostriches cannot fly.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
```



```
obj_spr.flight()
```

```
obj_ost.intro()
```

```
obj_ost.flight()
```

Output

```
There are many types of birds.  
Most of the birds can fly but some cannot.  
There are many types of birds.  
Sparrows can fly.  
There are many types of birds.  
Ostriches cannot fly.
```

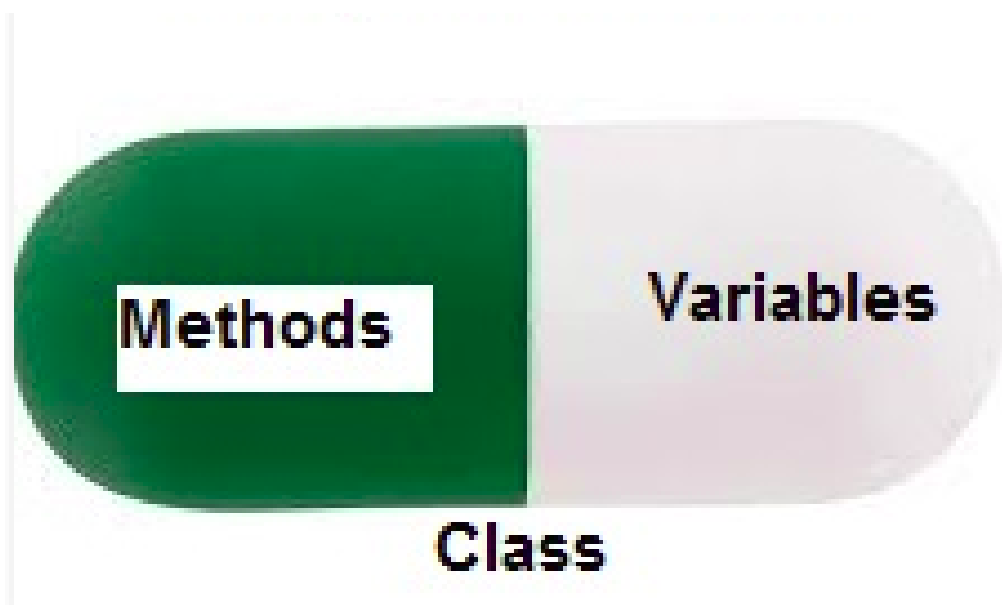
Note: For more information, refer to our [Polymorphism in Python](#) Tutorial.

Python Encapsulation

[Free Python 3 Tutorial](#)[Data Types](#)[Control Flow](#)[Functions](#)[List](#)[String](#)[Set](#)[Tuple](#)[Dictionary](#)[Oops](#)[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables.

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.



Encapsulation in Python

In the above example, we have created the `c` variable as the private attribute. We cannot even access this attribute directly and can't even change its value.

Python3

```
# Python program to
# demonstrate private members

# Creating a Base class
class Base:
    def __init__(self):
        self.a = "GeeksforGeeks"
        self.__c = "GeeksforGeeks"

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)


# Driver code
obj1 = Base()
print(obj1.a)

# Uncommenting print(obj1.c) will
# raise an AttributeError

# Uncommenting obj2 = Derived() will
# also raise an AttributeError as
# private member of base class
# is called inside derived class
```

Output

GeeksforGeeks

Note: for more information, refer to our [Encapsulation in Python](#) Tutorial.

Data Abstraction

It hides unnecessary code details from the user. Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction came.

Data Abstraction in Python can be achieved by creating abstract classes.

[Object Oriented Programming in Python | Set 2 \(Data Hiding and Object Printing\)](#).

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses](#) are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you. Don't miss out - [check it out now!](#)

Last Updated : 23 Aug, 2023

539

[Previous](#)

[Next](#)

Memoization using decorators in Python

Python Classes and Objects

Similar Reads

Shuffle a deck of card with OOPS in Python

Python OOPS - Aggregation and Composition

30 OOPs Interview Questions and Answers (2023)

Top 10 Advance Python Concepts That You Must Know

Word Prediction using concepts of N - grams and CDF

Important differences between Python 2.x and Python 3.x with examples

Python program to build flashcard using class in Python

Python | Sort Python Dictionaries by Key or Value

Python | Merge Python key values to list

Reading Python File-Like Objects from C | Python

Complete Tutorials

OpenAI Python API - Complete Guide

Pandas AI: The Generative AI Python Library

Python for Kids - Fun Tutorial to Learn Python Programming

Data Analysis Tutorial

Article Contributed By :



GeeksforGeeks

Vote for difficulty

Current difficulty : [Easy](#)

Easy

Normal

Medium

Hard

Expert

Improved By : [Kushagra vatsa](#), [nikhilaggarwal3](#), [raj095907](#), [tripathipriyanshu1998](#),
[surajkr_gupta](#), [anushk6zi0](#)

Article Tags : [python-oop-concepts](#) , [Python](#)

Practice Tags : [python](#)

Improve Article

Report Issue



GeeksforGeeks
Sanchhaya Education Private Limited

A-143, 9th Floor, Sovereign Corporate
Tower, Sector-136, Noida, Uttar Pradesh -
201305



Company

About Us
Legal
Careers
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
Apply for Mentor

Explore

Job-A-Thon
Hiring Challenge
Hack-A-Thon
GfG Weekly Contest
Offline Classes (Delhi/NCR)
DSA in JAVA/C++
Master System Design
Master CP
GeeksforGeeks Videos

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
Maths For Machine Learning
Pandas Tutorial
NumPy Tutorial
Deep Learning Tutorial

HTML & CSS

HTML
CSS
Bootstrap
Tailwind CSS
SASS
LESS
Web Design

Computer Science

GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

Python

Python
Programming Examples
Django Tutorial
Python Projects
Python Tkinter
OpenCV Python Tutorial
Python Interview Question

DevOps

Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

Competitive Programming

Top DS or Algo for CP
Top 50 Tree
Top 50 Graph
Top 50 Array
Top 50 String
Top 50 DP
Top 15 Websites for CP

System Design

What is System Design
Monolithic and Distributed SD
High Level Design or HLD
Low Level Design or LLD
Crack System Design Round
System Design Interview Questions
Grokking Modern System Design

JavaScript

TypeScript
ReactJS
NextJS
AngularJS
NodeJS
Express.js
Lodash
Web Browser

NCERT Solutions	School Subjects	Commerce	Management & Finance	UPSC	SSC/ BANKING
NCERT Solutions for Class 12	Mathematics	Accountancy	Management	Polity Notes	SSC CGL Syllabus
NCERT Solution for Class 11	Physics	Business Studies	HR Managment	Geography Notes	SBI PO Syllabus
NCERT Solutions for Class 10	Chemistry	Indian Economics	Income Tax	History Notes	SBI Clerk Syllabus
NCERT Solutions for Class 9	Biology	Macroeconomics	Finance	Science and Technology Notes	IBPS PO Syllabus
NCERT Solutions for Class 8	Social Science	Microeconimics	Economics	Economics Notes	IBPS Clerk Syllabus
Complete Study Material	English Grammar	Statistics for Economics		Important Topics in Ethics	SSC CGL Practice Papers
				UPSC Previous Year Papers	
Colleges	Companies	Preparation Corner	Exams	More Tutorials	Write & Earn
Indian Colleges Admission & Campus Experiences	IT Companies		JEE Mains		Write an Article
Top Engineering Colleges	Software Development Companies	Company Wise Preparation	JEE Advanced	Software Testing	Improve an Article
Top BCA Colleges	Artificial Intelligence(AI) Companies	Preparation for SDE	GATE CS	Software Development	Pick Topics to Write
Top MBA Colleges	CyberSecurity Companies	Experienced Interviews	NEET	Product Management	Share your Experiences
Top Architecture College	Service Based Companies	Internship Interviews	UGC NET	SAP	Internships
Choose College For Graduation	Product Based Companies	Competitive Programming	CAT	SEO	
	PSUs for CS Engineers	Aptitude Preparation		Linux	
		Puzzles		Excel	