# Python Tutorial - Learn Python Programming Language

**Last Updated :**05 Jul, 2025

Python is one of the most popular programming languages. It’s simple to use, packed with features and supported by a wide range of libraries and frameworks. Its clean syntax makes it beginner-friendly.It's

* A high-level language, used in web development, data science, automation, AI and more.
* Known for its readability, which means code is easier to write, understand and maintain.
* Backed by library support, so we don’t have to build everything from scratch, there’s probably a library that already does what we need.

**Why to Learn Python?**

* Requires fewer lines of code compared to other programming languages.
* Provides Libraries / Frameworks like Django, Flask, Pandas, Tensorflow, Scikit-learn and many more for Web Development, AI/ML, Data Science and Data Analysis
* Cross-platform, works on Windows, Mac and Linux without major changes.
* Used by top tech companies like Google, Netflix and NASA.
* Many Python coding job opportunities in Software Development, Data Science and AI/ML.

# Python OOPs Concepts

**Last Updated :**17 Mar, 2025

Object Oriented Programming is a fundamental concept in Python, empowering developers to build modular, maintainable, and scalable applications. By understanding the core OOP principles (classes, objects, inheritance, encapsulation, polymorphism, and abstraction), programmers can leverage the full potential of Python OOP capabilities to design elegant and efficient solutions to complex problems.

OOPs is a way of organizing code that uses objects and classes to represent real-world entities and their behavior. In OOPs, object has attributes thing that has specific data and can perform certain actions using methods.

**OOPs Concepts in Python**

* Class in Python
* Objects in Python
* Polymorphism in Python
* Encapsulation in Python
* Inheritance in Python
* Data Abstraction in Python

**Python Class**

A class is a collection of objects. [Classes](https://www.geeksforgeeks.org/python-classes-and-objects/)are blueprints for creating objects. A class defines a set of attributes and methods that the created objects (instances) can have.

**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. Example: Myclass.Myattribute

**Python Objects**

An Object is an instance of a Class. It represents a specific implementation of the class and holds its own data.

An object consists of:

* **State:** It is represented by the attributes and reflects the properties of an object.
* **Behavior:** It is represented by the methods of an object and 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.

### Creating Object

Creating an object in Python involves instantiating a class to create a new instance of that class. This process is also referred to as object instantiation.



class Dog:

species = "Canine" # Class attribute

​

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

self.name = name # Instance attribute

self.age = age # Instance attribute

​

# Creating an object of the Dog class

dog1 = Dog("Buddy", 3)

​

print(dog1.name)

print(dog1.species)

**Output**

Buddy

**Explanation:**

* **self.name:**Refers to the name attribute of the object (dog1) calling the method.
* **dog1.bark():**Calls the bark method on dog1.

***Note:*** *For more information, refer to* [*self in the Python class*](https://www.geeksforgeeks.org/self-in-python-class/)

### \_\_init\_\_ Method

[\_\_init\_\_](https://www.geeksforgeeks.org/__init__-in-python/) method is the constructor in Python, automatically called when a new object is created. It initializes the attributes of the class.

**Explanation:**

* **\_\_init\_\_:**Special method used for initialization.
* **self.name and self.age:** Instance attributes initialized in the constructor.

### Class and Instance Variables

In Python, variables defined in a class can be either class variables or instance variables, and understanding the distinction between them is crucial for object-oriented programming.

**Class Variables**

These are the variables that are shared across all instances of a class. It is defined at the class level, outside any methods. All objects of the class share the same value for a class variable unless explicitly overridden in an object.

**Instance Variables**

Variables that are unique to each instance (object) of a class. These are defined within the \_\_init\_\_ method or other instance methods. Each object maintains its own copy of instance variables, independent of other objects.

**Explanation:**

* **Class Variable (species):**Shared by all instances of the class. Changing Dog.species affects all objects, as it's a property of the class itself.
* **Instance Variables (name, age):** Defined in the \_\_init\_\_ method. Unique to each instance (e.g., dog1.name and dog2.name are different).
* **Accessing Variables:** Class variables can be accessed via the class name (Dog.species) or an object (dog1.species). Instance variables are accessed via the object (dog1.name).
* **Updating Variables:**Changing Dog.species affects all instances. Changing dog1.name only affects dog1 and does not impact dog2.

**Python Inheritance**

Inheritance allows a class (child class) to acquire properties and methods of another class (parent class). It supports hierarchical classification and promotes code reuse.

### ****Types of Inheritance:****

1. **Single Inheritance:** A child class inherits from a single parent class.
2. **Multiple Inheritance:**A child class inherits from more than one parent class.
3. **Multilevel Inheritance:** A child class inherits from a parent class, which in turn inherits from another class.
4. **Hierarchical Inheritance:** Multiple child classes inherit from a single parent class.
5. **Hybrid Inheritance:** A combination of two or more types of inheritance.

## Python Polymorphism

Polymorphism allows methods to have the same name but behave differently based on the object's context. It can be achieved through method overriding or overloading.

### Types of Polymorphism

1. **Compile-Time Polymorphism**: This type of polymorphism is determined during the compilation of the program. It allows methods or operators with the same name to behave differently based on their input parameters or usage. It is commonly referred to as method or operator overloading.
2. **Run-Time Polymorphism**: This type of polymorphism is determined during the execution of the program. It occurs when a subclass provides a specific implementation for a method already defined in its parent class, commonly known as method overriding.