# Object Oriented Programming

Object-oriented programming (OOP) is a programming paradigm that organizes data and behavior into reusable structures called objects. Python is an object-oriented programming language that provides support for creating classes, which are the blueprints for creating objects. Here's an introduction to the basics of OOP and classes in Python:

## Classes

A class is a user-defined data type that encapsulates data (attributes) and methods (functions) that operate on that data. It serves as a blueprint for creating objects. In Python, you define a class using the class keyword.

Example:

```
class MyClass:
    pass  # The 'pass' statement is used when you have an empty class
definition
```

## Objects

An object is an instance of a class. When you create an object, it has its own set of attributes and can execute methods defined in the class.

Example:

```
obj = MyClass()  # Creating an object of the class MyClass
```

## Attributes

Attributes are variables that belong to an object or class. They store data associated with the objects. There are two types of attributes: instance attributes and class attributes.

Instance attributes are specific to each object and are defined within methods using the self parameter. Class attributes are shared by all instances of the class and are defined directly within the class. Example:

```
class MyClass:
    class_attribute = "Shared attribute"  # Class attribute

    def __init__(self):
        self.instance_attribute = "Instance attribute"  # Instance
attribute
```

## Methods

Methods are functions defined within a class that can operate on the attributes of an object. They provide behavior to the objects.

Example:

```
class MyClass:
    def __init__(self):
        self.instance_attribute = "Instance attribute"

    def instance_method(self):
        print("This is an instance method.")

obj = MyClass()
obj.instance_method()  # Calling an instance method
```

## Inheritance

Inheritance allows creating a new class (derived class) based on an existing class (base class). The derived class inherits the attributes and methods of the base class and can also add its own.

Example:

```
class BaseClass:
    def base_method(self):
        print("This is a base method.")

class DerivedClass(BaseClass):
    def derived_method(self):
        print("This is a derived method.")

obj = DerivedClass()
obj.base_method()  # Calling a method from the base class
obj.derived_method()  # Calling a method from the derived class
```

These are some of the basic concepts of object-oriented programming and classes in Python. By utilizing these concepts, you can create modular and reusable code structures to build complex applications.