# Python Inheritance

Being an object-oriented language, Python supports class inheritance. It allows us to create a new class from an existing one.

- The newly created class is known as the **subclass** (child or derived class).
- The existing class from which the child class inherits is known as the superclass (parent or base class).

---

## Python Inheritance Syntax

```
# define a superclass
class super_class:
    # attributes and method definition

# inheritance
class sub_class(super_class):
    # attributes and method of super_class
    # attributes and method of sub_class
```

Here, we are inheriting the `sub_class` from the `super_class`.

**Note**: Before you move forward with inheritance, make sure you know how Python classes and objects work.

---

### Example: Python Inheritance

```
class Animal:

    # attribute and method of the parent class
    name = ""

    def eat(self):
        print("I can eat")

# inherit from Animal
class Dog(Animal):

    # new method in subclass
    def display(self):
        # access name attribute of superclass using self
        print("My name is ", self.name)

# create an object of the subclass
labrador = Dog()

# access superclass attribute and method
labrador.name = "Rohu"
labrador.eat()

# call subclass method
labrador.display()
```

**Output**

```
I can eat
My name is  Rohu
```

In the above example, we have derived a subclass *Dog* from a superclass *Animal*. Notice the statements,

```
labrador.name = "Rohu"

labrador.eat()
```

Here, we are using *labrador* (object of *Dog*) to access *name* and `eat()` of the *Animal* class.

This is possible because the subclass inherits all attributes and methods of the superclass.

Also, we have accessed the *name* attribute inside the method of the *Dog* class using `self`.

Python Inheritance Implementation

---

## is-a relationship

Inheritance is an **is-a** relationship. That is, we use inheritance only if there exists an **is-a** relationship between two classes. For example,

- **Car** is a **Vehicle**
- **Apple** is a **Fruit**
- **Cat** is an **Animal**

Here, **Car** can inherit from **Vehicle**, **Apple** can inherit from **Fruit**, and so on.

---

# Method Overriding in Python Inheritance

In the previous example, we see the object of the subclass can access the method of the superclass.

**However, what if the same method is present in both the superclass and subclass?**

In this case, the method in the subclass overrides the method in the superclass. This concept is known as method overriding in Python.

## Example: Method Overriding

```
class Animal:

    # attributes and method of the parent class
    name = ""

    def eat(self):
        print("I can eat")

# inherit from Animal
class Dog(Animal):

    # override eat() method
    def eat(self):
        print("I like to eat bones")

# create an object of the subclass
labrador = Dog()

# call the eat() method on the labrador object
labrador.eat()
```

**Output**

```
I like to eat bones
```

In the above example, the same method `eat()` is present in both the *Dog* class and the *Animal* class.

Now, when we call the `eat()` method using the object of the *Dog* subclass, the method of the *Dog* class is called.

This is because the `eat()` method of the *Dog* subclass overrides the same method of the *Animal* superclass.

---

# The super() Function in Inheritance

Previously we saw that the same method (function) in the subclass overrides the method in the superclass.

However, if we need to access the superclass method from the subclass, we use the `super()` function. For example,

```
class Animal:

    name = ""

    def eat(self):
        print("I can eat")

# inherit from Animal
class Dog(Animal):

    # override eat() method
    def eat(self):

        # call the eat() method of the superclass using super()
        super().eat()

        print("I like to eat bones")

# create an object of the subclass
labrador = Dog()

labrador.eat()
```

**Output**

```
I can eat
I like to eat bones
```

In the above example, the `eat()` method of the *Dog* subclass overrides the same method of the *Animal* superclass.

Inside the *Dog* class, we have used

```
# call method of superclass
super().eat()
```

to call the `eat()` method of the *Animal* superclass from the *Dog* subclass.

So, when we call the `eat()` method using the *labrador* object

```
# call the eat() method
labrador.eat()
```

Both the overridden and the superclass version of the `eat()` method is executed.

To learn more, visit Python super().

---

## More on Python Inheritance

Inheritance Types

There are 5 different types of inheritance in Python. They are:

- **Single Inheritance**: a child class inherits from only one parent class.
- **Multiple Inheritance**: a child class inherits from multiple parent classes.
- **Multilevel Inheritance**: a child class inherits from its parent class, which is inheriting from its parent class.
- **Hierarchical Inheritance**: more than one child class are created from a single parent class.
- **Hybrid Inheritance**: combines more than one form of inheritance.

Uses of Inheritance

- **Code Reusability**: Since a child class can inherit all the functionalities of the parent's class, this allows code reusability.
- **Efficient Development**: Once a functionality is developed, we can simply inherit it which allows for cleaner code and easy maintenance.
- **Customization**: Since we can also add our own functionalities in the child class, we can inherit only the useful functionalities and define other required features.

---

**Also Read:**

- Python Object Oriented Programming
- Python Multiple Inheritance

### Table of Contents