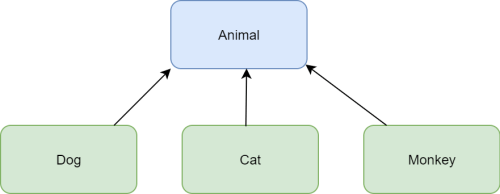
**Inheritance:**

Inheritance allows you to define a class in terms of another class, which makes it easier to create and maintain an application. To use inheritance, you start with a base (parent) class and then derive a child class from the base class. Inheritance works under the “is a” relationship model.

What is unique about an “is a” relation is that it works only in one direction, which is a child to parent. The following diagram demonstrates the “is a” relationship for some animals.

[](https://i2.wp.com/www.brightdevelopers.com/wp-content/uploads/2017/07/cpp_inheritance_basics.png?ssl=1)

The base class is Animal and the derived classes (children) are Dog, Cat, and Monkey. Translating one of the relationships into a statement, you would get along the lines of “A monkey is an animal”. Notice that the other way around doesn’t work. The phrase, “An animal is a monkey” is not a true statement.

Inheritance is one of the most important aspects of Object Oriented Programming (OOP). The key to understanding Inheritance is that it provides code re-usability. In place of writing the same code, again and again, we can simply inherit the properties of one class into the other.

OOP is all about real-world objects and inheritance is a way of representing real-world relationships. Here’s an example – **car, bus, bike** – all of these come under a broader category called **Vehicle**. That means they’ve inherited the properties of class vehicles i.e all are used for transportation.

We can represent this relationship in code with the help of inheritance.

**What is Inheritance in Object Oriented Programming:**

Inheritance is the procedure in which one class inherits the attributes and methods of another class. The class whose properties and methods are inherited is known as the Parent class. And the class that inherits the properties from the parent class is the Child class.

**The interesting thing is, along with the inherited properties and methods, a child class can have its own properties and methods.**

You may use the following syntax:\ to implement inheritance in Python:

class parent\_class:

body of parent class

class child\_class( parent\_class):

body of child class

Let’s see the implementation:

class Car: #parent class

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

       self.name = name

        self.mileage = mileage

   def description(self):

        return f"The {self.name} car gives the mileage of {self.mileage}km/l"

class BMW(Car): #child class

pass

class Audi(Car): #child class

def audi\_desc(self):

return "This is the description method of class Audi."

obj1 = BMW("BMW 7-series",39.53)

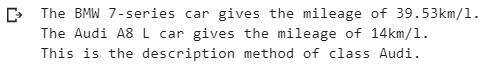
print(obj1.description())

obj2 = Audi("Audi A8 L",14)

print(obj2.description())

print(obj2.audi\_desc())

**Output:**



We have created two child classes namely “BMW” and “Audi” that have inherited the methods and properties of the parent class “Car”.  We have provided no additional features and methods in the class BMW. Whereas there is one additional method inside the class Audi.

Notice how the instance method description() of the parent class is accessible by the objects of child classes with the help of obj1.description() and obj2.description(). And the separate method of class Audi is also accessible using obj2.audi\_desc().

We can check the base or parent class of any class using a built-in class attribute **\_\_bases\_\_**

print(BMW.\_\_bases\_\_, Audi.\_\_bases\_\_)

Inheritance in Object Oriented Programming - Print Class

As we can here, the base class of both sub-classes is **Car.**Now, let’s see what happens when using \_\_base\_\_ with the parent class Car:

print( Car.\_\_bases\_\_ )

Inheritance in Object Oriented Programming - Print sub-class

Whenever we create a new class in Python 3.x, it is inherited from a built-in basic class called **Object.** In other words, the Object class is the root of all classes.

### Inheritance:

Inheritance allows classes to inherit features of other classes. Put another way, parent classes extend attributes and behaviors to child classes. **Inheritance supports reusability**.

If basic attributes and behaviors are defined in a parent class, child classes can be created extending the functionality of the parent class, and adding additional attributes and behaviors.

For example, herding dogs have the unique ability to herd animals. In other words, all herding dogs are dogs, but not all dogs are herding dogs. We represent this difference by creating a child class HerdingDog from the parent class Dog, and then add the unique herd() behavior.

The benefits of inheritance are programs can create a generic parent class, and then create more specific child classes as needed. This simplifies overall programming, because instead of recreating the structure of the Dog class multiple times, **child classes automatically gain access to functionalities within their parent class.**

In the following code snippet, child class HerdingDog inherits the method bark from the parent class Dog, and the child class adds an additional method, herd().

/Parent class Dog

class Dog{

    //Declare protected (private) fields

    \_attendance = 0;

    constructor(namee, birthday) {

        this.name = name;

        this.birthday = birthday;

    }

    getAge() {

        //Getter

        return this.calcAge();

    }

    calcAge() {

        //calculate age using today's date and birthday

        return this.calcAge();

    }

    bark() {

        return console.log("Woof!");

    }

    updateAttendance() {

        //add a day to the dog's attendance days at the petsitters

        this.\_attendance++;

}

}

//Child class HerdingDog, inherits from parent Dog

class HerdingDog extends Dog {

    constructor(name, birthday) {

        super(name);

        super(birthday);

    }

    herd() {

        //additional method for HerdingDog child class

        return console.log("Stay together!")

    }

}

Notice that the HerdingDog class does not have a copy of the bark() method, it inherits the bark() method defined in the parent Dog class.

When the code calls fluffy.bark() method, the bark() method walks up the chain of child to parent classes, to find where the bark method is defined.

//Parent class Dog

class Dog{

    //Declare protected (private) fields

    \_attendance = 0;

    constructor(namee, birthday) {

        this.name = name;

        this.birthday = birthday;

    }

    getAge() {

        //Getter

        return this.calcAge();

    }

    calcAge() {

        //calculate age using today's date and birthday

        return this.calcAge();

    }

    bark() {

        return console.log("Woof!");

    }

    updateAttendance() {

        //add a day to the dog's attendance days at the petsitters

        this.\_attendance++;

}

}

//Child class HerdingDog, inherits from parent Dog

class HerdingDog extends Dog {

    constructor(name, birthday) {

        super(name);

        super(birthday);

    }

    herd() {

        //additional method for HerdingDog child class

        return console.log("Stay together!")

    }

}

//instantiate a new HerdingDog object

const fluffy = new HerdingDog("Fluffy", "1/12/2019");

fluffy.bark();