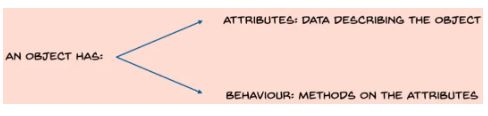
Python is a multi-paradigm programming language. Meaning, it supports different programming approaches. One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming ([OOP](https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/)).

[Java](https://www.simplilearn.com/java-tutorial) and [Scala](https://www.simplilearn.com/introduction-to-programming-in-scala-tutorial-video) are also OOP languages. At their core, every instance in these languages is an object, and every object has **attributes** and **behavior**.

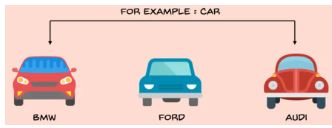


**Python Attributes**

Let’s say we have an object: a BMW car. Attributes of this car include the year of manufacture, maximum speed, the number of airbags, etc. The behavior of the car could be the movement of the speedometer, change in speed, etc.

**Python OOP Classes**

A **class** is a blueprint for the object. Each object is an instance of some class. It is a collection of similar objects.



PyCharm IDE is used here to implement the code. Learn how to install PyCharm and code with it in the [Simplilearn PyCharm tutorial](https://www.youtube.com/watch?v=hK9-_s55CBY). In the video, you’ll learn how to create new projects, add files, and open existing projects. You'll also learn how to modify the look of the UI and set the interpreter, and run a few codes on Pycharm to visualize plots, data frames, and the stored variables.

Below is a simple example of how to define a class and create an instance of that class.

**class** car: *#Defining a class*

**def** getSpeed(self):

       print(**"155 mph"**)

BMW = car()*#Creating an instance of class*

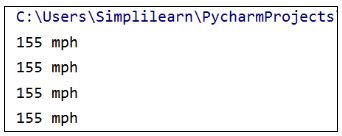
FORD = car()

car.getSpeed(BMW)

car.getSpeed(FORD)

BMW.getSpeed()

FORD.getSpeed()



**Self**

For creating an instance of any class using Python Object Oriented programming, use braces (curly brackets) along with the class name. In this case, the **self** is taken as a reference variable, which in turn points to the current class object.

As a rule, **self** is the first argument in the function definition. However, usage of **self** is optional in case of the function call. All classes create objects, and subsequently, all objects contain within them, characteristics referred to as attributes.

**\_\_init\_\_()**

We use the **\_\_init\_\_()** method in Python Object Oriented Programming language to initialize the initial attributes of an object by assigning them their default state or value. This method necessarily needs at least one argument, along with the **self** variable pointing to the object itself.

**class** car:

**def** \_\_init\_\_(self, year, speed):

       self.year=year

       self.speed=speed

**def** getSpeed(self):

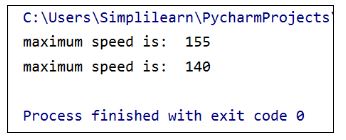
       print(**"maximum speed is: "**,self.speed)

BMW = car(2018, 155)

FORD = car(2016, 140)

BMW.getSpeed()

FORD.getSpeed()



**Encapsulation**

A key aspect of Python object oriented programming is [encapsulation](https://www.geeksforgeeks.org/encapsulation-in-python/#:~:targetText=Encapsulation%20in%20Python,the%20accidental%20modification%20of%20data.), used for restricting access to variables and methods.

In encapsulation, the code and data are wrapped together within a single unit from being modified by accident. In the code below, all the functions defined in the class are encapsulated and cannot be changed.

**class** car:

**def** \_\_init\_\_(self, year, speed):

      self.year=year

      self.speed=speed

**def** getSpeed(self):

      print(**"maximum speed is: "**,self.speed)

**def** setSpeed(self,speed):

      self.speed=speed

BMW = car(2018, 155)

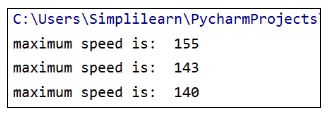
FORD = car(2016, 140)

BMW.getSpeed()

BMW.setSpeed(143)

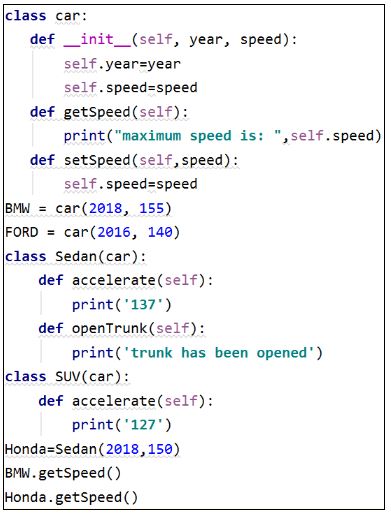
BMW.getSpeed()

FORD.getSpeed()



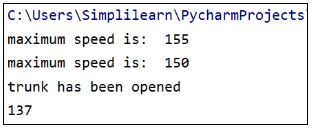
**Inheritance**

**Inheritance** is a critical feature of Python OOP language. It refers to defining a new class with little or no modification to an existing class. The new class is a **derived** (or child) class and the one from which it inherits is called the base (or parent) class.



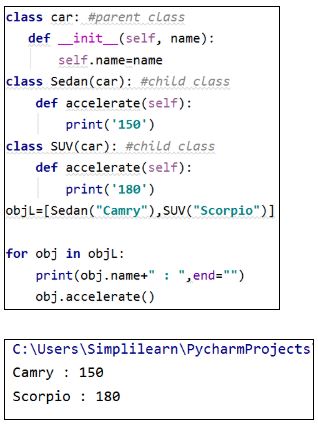
class char 1

In this case, Sedan and SUV are the two child classes that are inheriting the properties of the parent class.



**Polymorphism**

Polymorphism is a feature in Python object oriented programming that lets you use the same function in multiple ways.



In the above example, we have created an instance of both child classes looping through the objects. By doing this, we polymorphed the definition to do two different things depending on what our child class is.

Want to get skilled in working with Python classes and files? Then check out the [Python Training Course](https://www.simplilearn.com/mobile-and-software-development/python-development-training?source=GhPreviewCTAText#/course-preview). Click to enroll now!

**Master Python Today**

This tutorial demonstrates how to do OOP in Python object oriented programming, covering the concept of objects and classes, and how to define and create them. You should by now have an idea about the different features of OOP such as inheritance, encapsulation, and polymorphism.

Our [Python Training Course](https://www.simplilearn.com/mobile-and-software-development/python-development-training) will empower you to:

1. Internalize the concepts and constructs of Python
2. Learn to create your own Python programs
3. Master Python Django and advanced web development in Python
4. Master PyGame and game development in Python
5. Create a flappy bird game clone

Go through this Simplilearn video on “Python OOPS Concepts” delivered by our Software Development experts that covers the concepts of object and classes, and how to define and create them.