What Is A Python Class?

A class is an arrangement of variables and functions into a single logical entity. It works as a template for creating objects. Every object can use class variables and functions as its members.

Python has a reserved keyword known as “class” which you can use to define a new class.

The object is a working instance of a class created at runtime.

How To Create A Class In Python?

There are some terms which you need to know while working with classes in Python.

1. The “class” keyword  
2. The instance attributes  
3. The class attributes  
4. The “self” keyword  
5. The “\_\_init\_” method

Let’s now have a clear understanding of each of the above points one by one.

The “Class” Keyword

With the class keyword, we can create a Python class as shown in the example below.

class BookStore:

pass

What Is Self?

Python provides the “self” keyword to represent the instance of a class. It works as a handle for accessing the class members such as attributes from the class methods.

Also, please note that it is implicitly the first argument to the \_\_init\_\_ method in every Python class. You can read about it below.

What Is \_\_init\_\_ (Constructor) In Python?

The “\_\_init\_\_()” is a unique method associated with every Python class.

Python calls it automatically for every object created from the class. Its purpose is to initialize the class attributes with user-supplied values.

class BookStore:

def \_\_init\_\_(self):

print("\_\_init\_\_() constructor gets called...")

B1 = BookStore()

Output

\_\_init\_\_() constructor gets called...

The Instance Attributes

These are object-specific attributes defined as parameters to the \_\_init\_\_ method. Each object can have different values for themselves.

In the below example, the “attrib1” and “attrib2” are the instance attributes.

class BookStore:

def \_\_init\_\_(self, attrib1, attrib2):

self.attrib1 = attrib1

self.attrib2 = attrib2

The Class Attributes

Unlike the instance attributes which are visible at object-level, the class attributes remain the same for all objects.

Check out the below example to demonstrate the usage of class-level attributes.

class BookStore:

instances = 0

def \_\_init\_\_(self, attrib1, attrib2):

self.attrib1 = attrib1

self.attrib2 = attrib2

BookStore.instances += 1

b1 = BookStore("", "")

b2 = BookStore("", "")

print("BookStore.instances:", BookStore.instances)

In this example, the “instances” is a class-level attribute. You can access it using the class name. It holds the total no. of instances created.

We’ve created two instances of the class <Bookstore>. Hence, executing the example should print “2” as the output.

# output

BookStore.instances: 2

Python Class Demo

Given here is an example where we are building a BookStore class and instantiating its object with different values.

Create A BookStore Class In Python

class BookStore:

noOfBooks = 0

def \_\_init\_\_(self, title, author):

self.title = title

self.author = author

BookStore.noOfBooks += 1

def bookInfo(self):

print("Book title:", self.title)

print("Book author:", self.author,"\n")

# Create a virtual book store

b1 = BookStore("Great Expectations", "Charles Dickens")

b2 = BookStore("War and Peace", "Leo Tolstoy")

b3 = BookStore("Middlemarch", "George Eliot")

# call member functions for each object

b1.bookInfo()

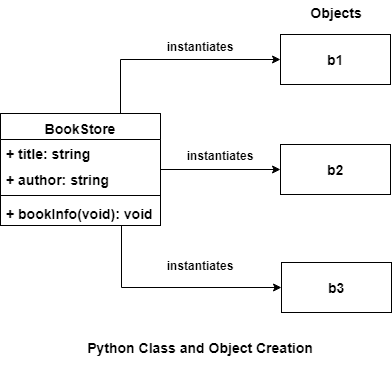
b2.bookInfo()

b3.bookInfo()

print("BookStore.noOfBooks:", BookStore.noOfBooks)

UML Diagram Of BookStore Class

The UML diagram of the above code is as follows.



Python Class and Objects (UML diagram)

After executing the code in the example, you should see the following result.

# output

Book title: Great Expectations

Book author: Charles Dickens

Book title: War and Peace

Book author: Leo Tolstoy

Book title: Middlemarch

Book author: George Eliot

BookStore.noOfBooks: 3