Contents

[1. Concepts of object-oriented programming 2](#_Toc172539599)

[2. Classes and Objects 3](#_Toc172539600)

# Concepts of object-oriented programming

Object-oriented programming revolves around defining and using new types.

In python a class is how python represents a type. A function isinstance() reports whether an object is an instance of a class. i.e whether an object has a particular type.

* **isinstance('abc', str)**
* **True**
* **isinstance(55.2, str)**
* **False**
* **isinstance('abc', object)**
* **True**
* **isinstance(max, object)**
* **True**

Python has a class called object. Every other class is based on it. Even classes and functions are instances of object.

OOP Concepts:

Class: A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods). A class is a user-defined data type that serves as a template for creating objects. It defines a set of attributes and methods that the created objects can use.

Object: An object is an instance of a class. It is a self-contained component that contains attributes and methods needed to make a particular type of data useful. When a class is defined, no memory is allocated until an object of that class is created. Objects are individual instances of a class that can have different values for the attributes defined in the class.

Attributes: Attributes are the variables that belong to an object. They are used to store the state of an object.

Methods: Methods are functions that belong to a class and define the behaviors of the objects created from the class.

Constructor: A constructor is a special method that is automatically called when an object of a class is created. It initializes the attributes of the object.

Inheritance: Inheritance is a mechanism where a new class (child class) inherits the attributes and methods of an existing class (parent class). This allows for code reuse and the creation of a hierarchical relationship between classes.

Encapsulation: Encapsulation is the bundling of data (attributes) and methods that operate on the data into a single unit, or class. It restricts direct access to some of an object's components, which can help prevent the accidental modification of data.

Abstraction: Abstraction is the concept of hiding the complex implementation details and showing only the essential features of the object. It helps in reducing programming complexity and effort.

Polymorphism: Polymorphism allows objects of different classes to be treated as objects of a common super class. It is typically used to call methods that behave differently depending on the object that invokes the method.

Overriding: Overriding occurs when a child class provides a specific implementation of a method that is already defined in its parent class.

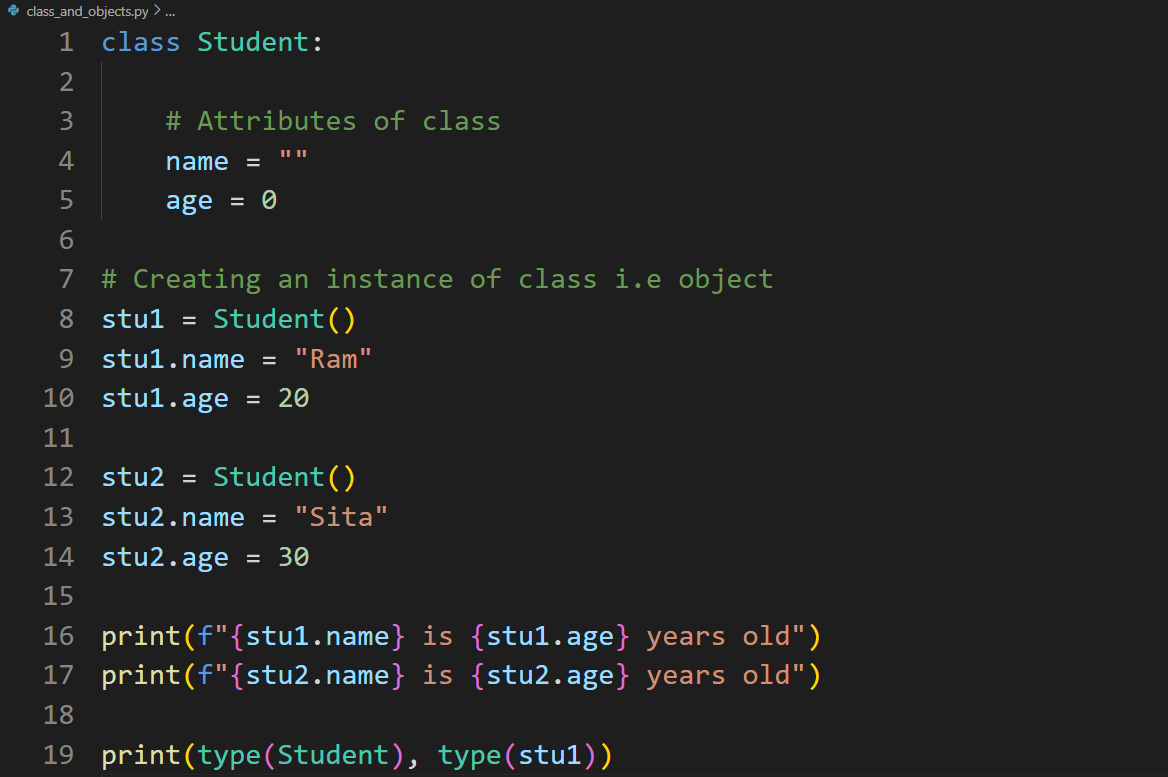
Overloading: Overloading is a feature that allows a class to have more than one method having the same name, if their parameter lists are different.

# Classes and Objects

Syntax of class:

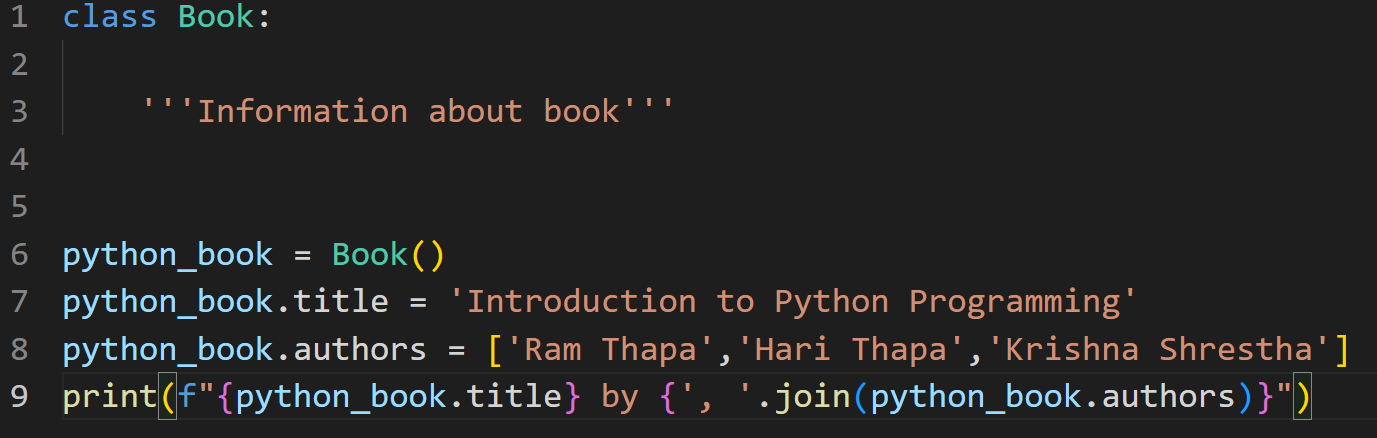
class ClassName:

# body of class



The given code defines a Student class with class-level attributes name and age, and then creates two instances of this class (stu1 and stu2) where the attributes are individually set for each instance. It prints out the name and age for each instance, demonstrating that these attributes can hold different values for different instances. The code also prints the types of the Student class and the stu1 instance, showing <class 'type'> for the class itself and <class '\_\_main\_\_.Student'> for the instance, indicating that Student is a class (created using the type metaclass) and stu1 is an instance of Student.

Try the following:

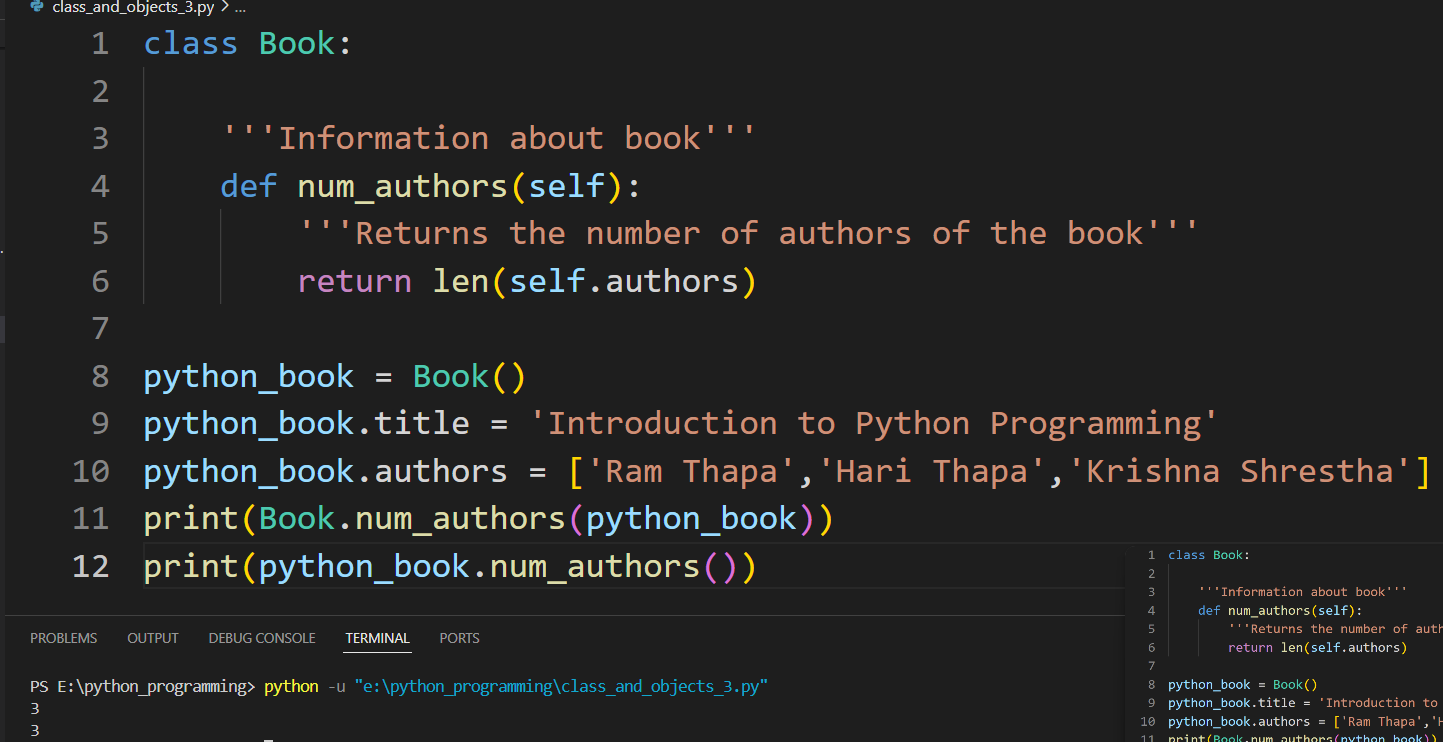
* print(isinstance(stu1, Student))
* print(isinstance(stu1, object))

The first assignment statement creates a Book object and then assigns that object to variable python\_book. The second assignment statement creates a title variable inside the Book object; that variable refers to the string 'Introduction to Python Programming'. The third assignment statement creates variable authors, also inside the Book object, which refers to the list of strings ['Ram Thapa','Hari Thapa','Krishna Shrestha'].

Variables title and authors are called instance variables because they are variables inside an instance of a class. We can access these instance variables through variable python\_book.

**Writing a Method in Class Book:**

We can define a method num\_authors that will return the numbers of authors of the book. Book method num\_authors looks just like a function except that it has a parameter called self, which refers to a Book. there are two ways to call a method. One way is to access the method through the class, and the other is to use object-oriented syntax.

* Book.num\_authors(python\_book)
* Python\_book.num\_authors()

Let’s take a close look at the first call on method num\_authors.

* Book.num\_authors(python\_book)

In class Book is method num\_authors. The argument to the call, python\_book, is passed to

parameter self. Python treats the second call on num\_authors exactly as it did the first; the first call

is equivalent to this one

* Python\_book.num\_authors()

The second version is much more common because it lists the object first; we think of that version as asking the book how many authors it has. Thinking of method calls this way can really help develop an object-oriented mentality.

In the python\_book example, we assigned the title and list of authors after the Book object was created. That approach isn’t scalable; we don’t want to have to type those extra assignment statements every time we create a Book. Instead, we’ll write a method that does this for us as we create the Book. This is a special method and is called \_\_init\_\_. We’ll also include the publisher, ISBN, and price as parameters of \_\_init\_\_:

|  |
| --- |
| class Book:  """Information about a book, including title, list of authors publisher, ISBN, and price.  """  def \_\_init\_\_(self, title, authors, publisher, isbn, price):  """Create a new book entitled title, written by the people in authors,  published by publisher, with ISBN isbn and costing price dollars.  >>> python\_book = Book( \  'Practical Programming', \  ['Campbell', 'Gries', 'Montojo'], \  'Pragmatic Bookshelf', \  '978-1-6805026-8-8', \  25.0)  """  self.title = title  self.authors = authors[:]  self.publisher = publisher  self.ISBN = isbn  self.price = price |
| def num\_authors(self):  '''Returns the number of authors of the book'''  return len(self.authors)  python\_book = Book(  'Practical Programming',  ['Campbell', 'Gries', 'Montojo'],  'Pragmatic Bookshelf',  '978-1-6805026-8-8',  25.0)  print(python\_book.ISBN, python\_book.title, python\_book.authors) |

Method \_\_init\_\_ is called whenever a Book object is created. Its purpose is to initialize the new object; this method is sometimes called a constructor. Here are the steps that Python follows when creating an object:

1. It creates an object at a particular memory address.

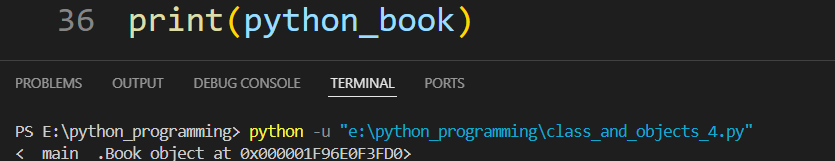
2. It calls method \_\_init\_\_, passing in the new object into the parameter self.

3. It produces that object’s memory address.

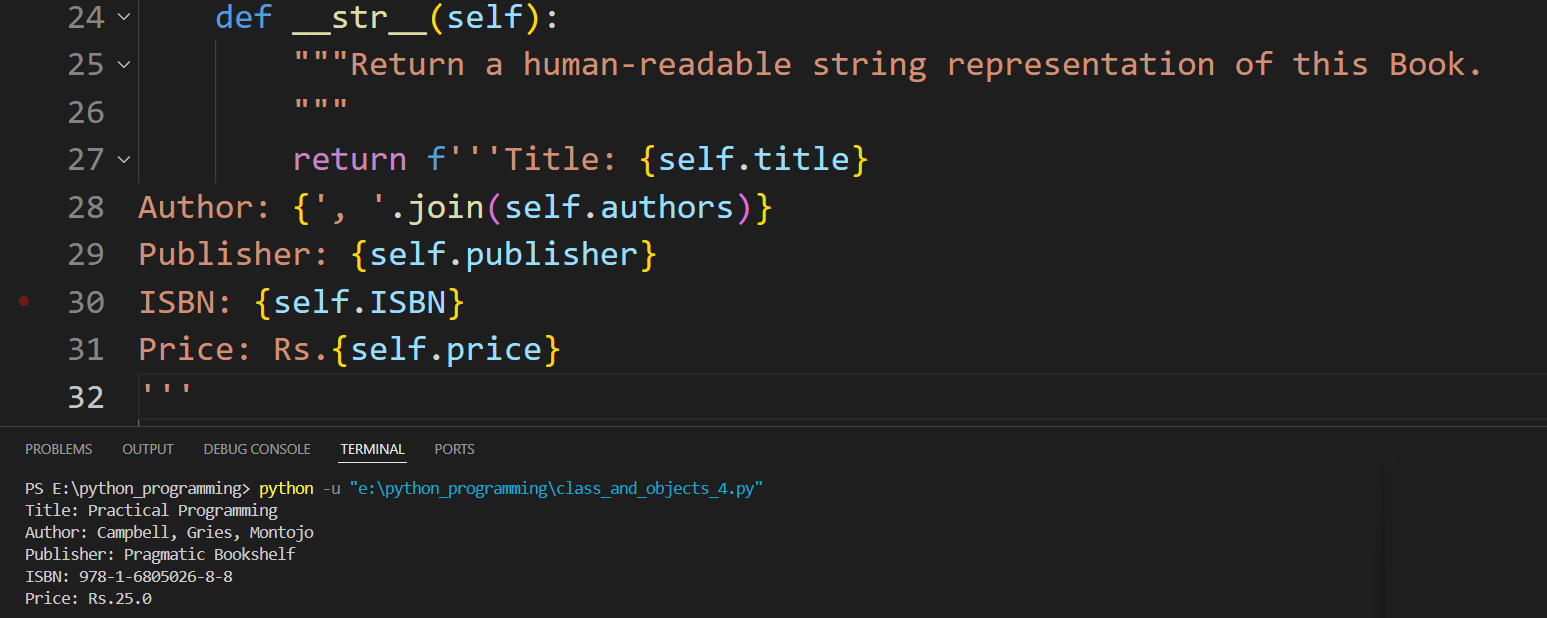
**Note: Methods belong to classes. Instance variables belong to objects. If we try to access**

**an instance variable as we do a method, we get an error.**

**Another special method: \_\_str\_\_()**

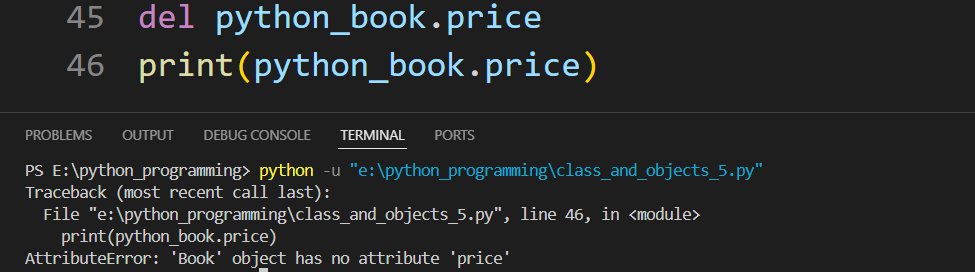
When we call print(python\_book), then python\_book.\_\_str\_\_() is called to find out what string to print. The output Python produces when we print a Book isn’t particularly useful:

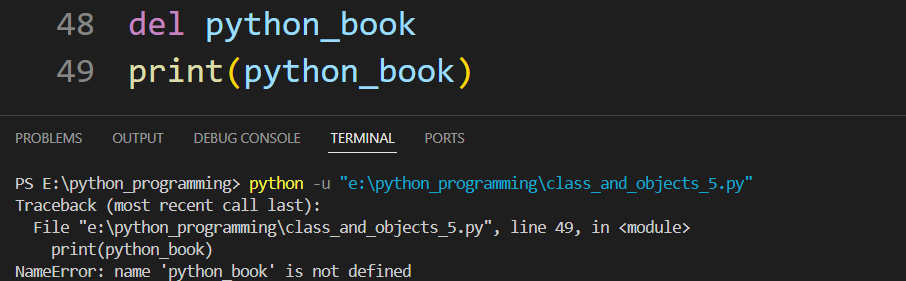
This is the default behavior for converting objects to strings: it just shows us where the object is in memory. This is the behavior defined in class object’s method \_\_str\_\_, which our Book class has inherited.

Let’s define method Book.\_\_str\_\_ to provide useful output; this method goes inside class Book, along with \_\_init\_\_ and num\_authors:

The result is displayed when print(python\_book)

**Deleting Object Properties**

****To delete an attribute from an object, you can use the del statement followed by the object's attribute.

**Deleting Objects**