Constructors in Python

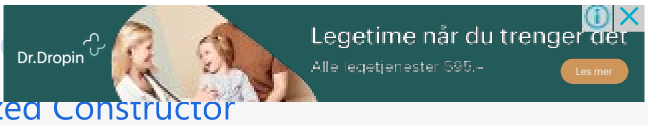
Updated on: August 28, 2021 | + 3 Comments

Constructor is a special method used to create and initialize an object of a class. On the other hand, a destructor is used to destroy the object.

After reading this article, you will learn:

- How to create a constructor to initialize an object in Python
- Different types of constructors
- Constructor overloading and chaining

Table of contents



What is Constructor in Python?

In object-oriented programming, **A constructor is a special method used to create and initialize an object of a class**. This method is defined in the class.

- The constructor is executed automatically at the time of object creation.
- The primary use of a constructor is to declare and initialize data member/ instance variables of a class. The constructor contains a collection of statements (i.e., instructions) that executes at the time of object creation to initialize the attributes of an object.

For example, when we execute `obj = Sample()`, Python gets to know that obj is an object of class `Sample` and calls the constructor of that class to create an object.

Note: In Python, internally, the `__new__` is the method that creates the object, and `__del__` method is called to destroy the object when the reference count for that object becomes zero.

In Python, Object creation is divided into two parts in **Object Creation** and **Object initialization**

- Internally, the `__new__` is the method that creates the object
- And, using the `__init__()` method we can implement constructor to initialize the object.

Syntax of a constructor

```python
def __init__(self):
    # body of the constructor
```

Where,

- `def`: The keyword is used to define function.
- `__init__()` Method: It is a reserved method. This method gets called as soon as an object of a class is instantiated.
- `self`: The first argument `self` refers to the current object. It binds the instance to the `__init__()` method. It's usually named `self` to follow the naming convention.

Note: The `__init__()` method arguments are optional. We can define a constructor with any number of arguments.

Example: Create a Constructor in Python

In this example, we'll create a Class **Student** with an instance variable student name. we'll see how to use a constructor to initialize the student name at the time of object creation.

```python
class Student:

    # constructor
    # initialize instance variable
    def __init__(self, name):
        print('Inside Constructor')
        self.name = name
        print('All variables initialized')

    # instance Method
    def show(self):
        print('Hello, my name is', self.name)


# create object using constructor
s1 = Student('Emma')
s1.show()
```

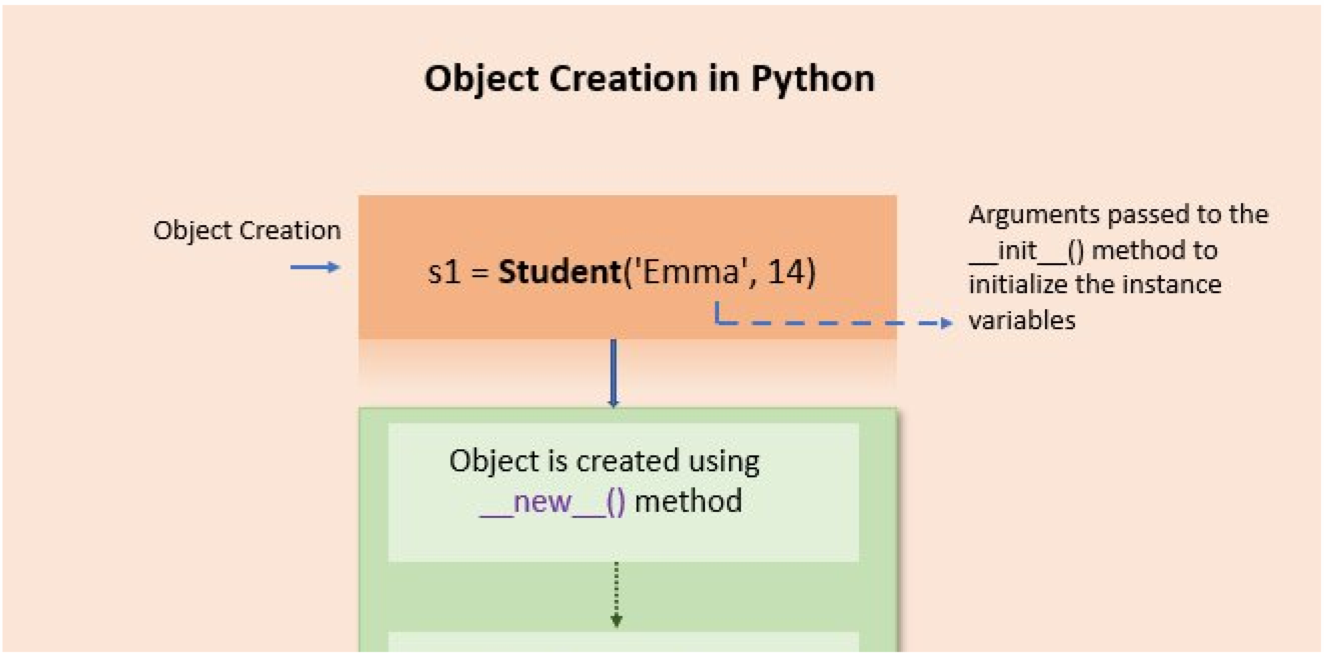
Run

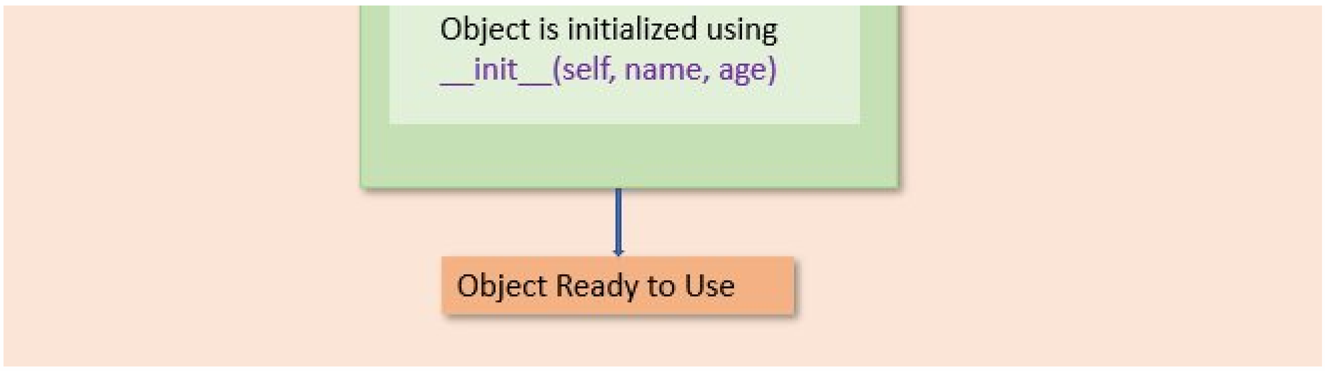
Output

```
Inside Constructor
All variables initialized

Hello, my name is Emma
```

- In the above example, an object `s1` is created using the constructor
- While creating a Student object `name` is passed as an argument to the `__init__()` method to initialize the object.
- Similarly, various objects of the Student class can be created by passing different names as arguments.


Object Creation in Python
Object Creation
s1 = Student('Emma', 14)
Arguments passed to the __init__() method to initialize the instance variables
Object is created using __new__() method



Create an object in Python using a constructor

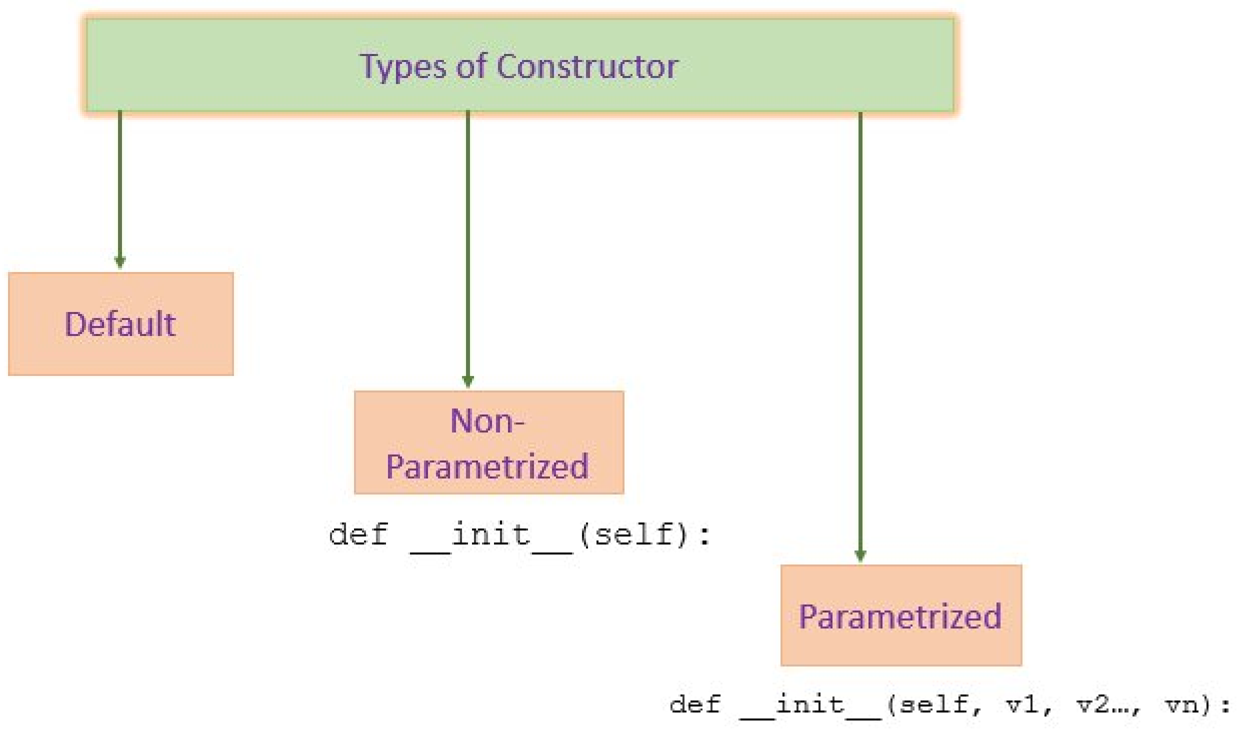
Note:

- For every object, the constructor will be executed only once. For example, if we create four objects, the constructor is called four times.
- In Python, every class has a constructor, but it’s not required to define it explicitly. Defining constructors in class is optional.
- Python will provide a default constructor if no constructor is defined.

Types of Constructors

In Python, we have the following three types of constructors.

- Default Constructor
- Non-parametrized constructor
- Parameterized constructor



Types of constructor

Default Constructor

Python will provide a default constructor if no constructor is defined. Python adds a default constructor when we do not include the constructor in the class or forget to declare it. It does not perform any task but initializes the objects. It is an empty constructor without a body.

If you do not implement any constructor in your class or forget to declare it, the Python inserts a default constructor into your code on your behalf. This constructor is known as the default constructor.

It does not perform any task but initializes the objects. It is an empty constructor without a body.

Note:

- The default constructor is not present in the source py file. It is inserted into the code during compilation if not exists. See the below image.
- If you implement your constructor, then the default constructor will not be added.

Example:

```
class Employee:

    def display(self):
        print('Inside Display')

emp = Employee()
emp.display()
```

Run

Output

```
Inside Display
```

As you can see in the example, we do not have a constructor, but we can still create an object for the class because Python added the default constructor during a program compilation.

Non-Parametrized Constructor

A constructor without any arguments is called a non-parameterized constructor. This type of constructor is used to initialize each object with default values.

This constructor doesn’t accept the arguments during object creation. Instead, it initializes every object with the same set of values.

```python
class Company:

    # no-argument constructor
    def __init__(self):
        self.name = "PYnative"
        self.address = "ABC Street"

    # a method for printing data members
    def show(self):
        print('Name:', self.name, 'Address:', self.address)

# creating object of the class
cmp = Company()

# calling the instance method using the object
cmp.show()
```

Run

Output

```
Name: PYnative Address: ABC Street
```

As you can see in the example, we do not send any argument to a constructor while creating an object.

Parameterized Constructor

A constructor with defined parameters or arguments is called a parameterized constructor. We can pass different values to each object at the time of creation using a parameterized constructor.

The first parameter to constructor is `self` that is a reference to the being constructed, and the rest of the arguments are provided by the programmer. A parameterized constructor can have any number of arguments.

For example, consider a company that contains thousands of employees. In this case, while creating each employee object, we need to pass a different name, age, and salary. In such cases, use the parameterized constructor.

Example:

```
class Employee:
    # parameterized constructor
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

    # display object
    def show(self):
        print(self.name, self.age, self.salary)

# creating object of the Employee class
emma = Employee('Emma', 23, 7500)
emma.show()

kelly = Employee('Kelly', 25, 8500)
kelly.show()
```

Run

Output

```
Emma 23 7500
Kelly 25 8500
```

In the above example, we define a parameterized constructor which takes three parameters.

Constructor With Default Values

Python allows us to define a constructor with default values. The default value will be used if we do not pass arguments to the constructor at the time of object creation.

The following example shows how to use the default values with the constructor.

Example

```
class Student:
    # constructor with default values age and classroom
    def __init__(self, name, age=12, classroom=7):
        self.name = name
        self.age = age
        self.classroom = classroom

    # display Student
    def show(self):
        print(self.name, self.age, self.classroom)

# creating object of the Student class
emma = Student('Emma')
emma.show()

kelly = Student('Kelly', 13)
kelly.show()
```

Run

Output

```
Emma 12 7
Kelly 13 7
```

As you can see, we didn’t pass the age and classroom value at the time of object creation, so default values are used.

Self Keyword in Python

As you all know, the class contains instance variables and methods. Whenever we define instance methods for a class, we use self as the first parameter. Using `self`, we can **access the instance variable and instance method** of the object.

The first argument `self` refers to the current object.

Whenever we call an instance method through an object, the Python compiler implicitly passes object reference as the first argument commonly known as self.

It is not mandatory to name the first parameter as a `self`. We can give any name whatever we like, but it has to be the first parameter of an instance method.

Example

```python
class Student:
    # constructor
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # self points to the current object
    def show(self):
        # access instance variable using self
        print(self.name, self.age)

# creating first object
emma = Student('Emma', 12)
emma.show()

# creating Second object
kelly = Student('Kelly', 13)
```

```
kelly.show()
```

Output

```
Emma 12
Kelly 13
```

Constructor Overloading

Constructor overloading is a concept of having more than one constructor with a different parameters list in such a way so that each constructor can perform different tasks.

For example, we can create a three constructor which accepts a different set of parameters

Python does not support constructor overloading. If we define multiple constructors then, the interpreter will considers only the last constructor and throws an error if the sequence of the arguments doesn’t match as per the last constructor. The following example shows the same.

Example

```
class Student:
    # one argument constructor
    def __init__(self, name):
        print("One arguments constructor")
        self.name = name

    # two argument constructor
    def __init__(self, name, age):
        print("Two arguments constructor")
        self.name = name
        self.age = age

# creating first object
emma = Student('Emma')

# creating Second object
kelly = Student('Kelly', 13)
```

Run

Output

```
TypeError: __init__() missing 1 required positional argument: 'age'
```

- As you can see in the above example, we defined multiple constructors with different arguments.
- At the time of object creation, the interpreter executed the second constructor because Python always considers the last constructor.
- Internally, the object of the class will always call the last constructor, even if the class has multiple constructors.
- In the example when we called a constructor only with one argument, we got a type error.

Constructor Chaining

Constructors are used for instantiating an object. The task of the constructor is to assign value to data members when an object of the class is created.

Constructor chaining is the process of calling one constructor from another constructor. Constructor chaining is useful when you want to invoke multiple constructors, one after another, by initializing only one instance.

In Python, **constructor chaining is convenient when we are dealing with inheritance**. When an instance of a child class is initialized, the constructors of all the parent classes are first invoked and then, in the end, the constructor of the child class is invoked.

Using the `super()` method we can invoke the parent class constructor from a child class.

Example

```python
class Vehicle:
    # Constructor of Vehicle
    def __init__(self, engine):
        print('Inside Vehicle Constructor')
        self.engine = engine

class Car(Vehicle):
    # Constructor of Car
    def __init__(self, engine, max_speed):
        super().__init__(engine)
        print('Inside Car Constructor')
        self.max_speed = max_speed

class Electric_Car(Car):
    # Constructor of Electric Car
    def __init__(self, engine, max_speed, km_range):
        super().__init__(engine, max_speed)
        print('Inside Electric Car Constructor')
        self.km_range = km_range

# Object of electric car
ev = Electric_Car('1500cc', 240, 750)
print(f'Engine={ev.engine}, Max Speed={ev.max_speed}, Km range={ev.km_rang
```

Run

Output

```
Inside Vehicle Constructor
Inside Car Constructor
Inside Electric Car Constructor

Engine=1500cc, Max Speed=240, Km range=750
```

Counting the Number of objects of a Class

The constructor executes when we create the object of the class. For every object, the constructor is called only once. So for counting the number of objects of a class, we can add a counter in the constructor, which increments by one after each object creation.

Example

```
class Employee:
    count = 0
    def __init__(self):
        Employee.count = Employee.count + 1
```

```
# creating objects
e1 = Employee()
e2 = Employee()
e2 = Employee()
print("The number of Employee:", Employee.count)
```

Run

Output

```
The number of employee: 3
```

Constructor Return Value

In Python, the constructor does not return any value. Therefore, while declaring a constructor, we don’t have anything like return type. Instead, a constructor is implicitly called at the time of object instantiation. Thus, it has the sole purpose of initializing the instance variables.

The `__init__()` is required to return None. We can not return something else. If we try to return a non-None value from the `__init__()` method, it will raise TypeError.

Example

```python
class Test:

    def __init__(self, i):
        self.id = i
        return True

d = Test(10)
```

Run

Output

```
TypeError: __init__() should return None, not 'bool'
```

Conclusion and Quick recap

In this lesson, we learned constructors and used them in object-oriented programming to design classes and create objects.

The below list contains the summary of the concepts we learned in this tutorial.

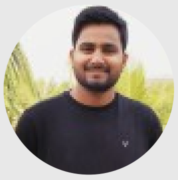
- A constructor is a unique method used to initialize an object of the class.
- Python will provide a default constructor if no constructor is defined.
- Constructor is not a method and doesn’t return anything. it returns None
- In Python, we have three types of constructor default, Non-parametrized, and

parameterized constructor.

- Using self, we can access the instance variable and instance method of the object. The first argument self refers to the current object.
- Constructor overloading is not possible in Python.
- If the parent class doesn't have a default constructor, then the compiler would not insert a default constructor in the child class.
- A child class constructor can also invoke the parent class constructor using the `super()` method.

Filed Under: Python, Python Object-Oriented Programming (OOP)

About Vishal



Founder of PYnative.com I am a Python developer and I love to write articles to help developers. Follow me on Twitter. All the best for your future Python endeavors!

Related Tutorial Topics:

Python Python Object-Oriented Programming (OOP)

Python Exercises and Quizzes

Free coding exercises and quizzes cover Python basics, data structure, data analytics, and more.

15+ **Topic-specific Exercises and Quizzes**
Each Exercise contains 10 questions
Each Quiz contains 12-15 MCQ

Exercises Quizzes

Posted In

Python Python Object-Oriented Programming (OOP)

Tweet F share in share P Pin

Python OOP

All Python Topics