codingem.com

Super() In Python: What Does It Do?

By Artturi Jalli

In Python, **super()** method makes it possible to access the members of a parent class.

To understand what is a parent class, you need to know what class inheritance means.

Before jumping into the details, let's see a quick example of using the **super()** method.

For instance:

```
class Pet():
    def hello(self): print("Hello, I am a Pet")

class Cat(Pet):
    def say(self):
        super().hello()

luna = Cat()
luna.say()
```

Output:

```
Hello, I am a Pet
```

In this piece of code:

- The **Pet** class defines a method called **hello()**.
- The **Cat** class inherits the **Pet** class.
- The **Cat** class implements a **say()** method that calls the **hello()** method from the **Pet** class.
- This is possible via the **super()** method.

In this guide, we are going to take a deeper look at how the **super()** method works in Python.

To truly understand **super()**, you need to understand what is inheritance in Python.

Class Inheritance in Python: A Quick Introduction

Python is an object-oriented programming language.

One of the main features of an object-oriented language like Python is class inheritance.

Python supports class inheritance, also called subclassing.

The idea behind the class inheritance is to create a parent class-child class hierarchy.

In other words, the members of the parent class are inherited to the child class.

To put it short, inheritance lets you reuse code.

But why reuse code?

You should always try to avoid repeating yourself in code.

Sometimes your Python objects are related to one another.

In this case, it would be useless if you had to repeat yourself and re-implement methods from one class to another class.

In this case, you can use inheritance.

A great example of inheritance would be a **Person**–**Student** relationship in the code.

- Lets's say your code has two classes, **Person** and **Student**.
- As you know, each **Student** is also a **Person**.
- Thus, it makes sense to inherit all the properties of a **Person** to an **Student**.

For example:

```
class Person:
    def __init__(self, name, age):
```

```
        self.name = name
        self.age = age

    def introduce(self):
        print(f"Hello, my name is {self.name}. I am {self.age} years old.")

class Student(Person):
    def __init__(self, name, age, graduation_year):
        # DO not worry about the next line, we will come back to it very soon!
        super().__init__(name, age)
        self.graduation_year = graduation_year

    def graduates(self):
        print(f"{self.name} will graduate in {self.graduation_year}")
```

Now you can use this setup to create **Student** objects that can use the **Person** classes **introduce()** method.

For instance:

```
alice = Student("Alice", 30, 2023)
alice.introduce()
alice.graduates()
```

Output:

```
Hello, my name is Alice. I am 30 years old.
Alice will graduate in 2023
```

In this piece of code:

- **alice.introuce()** is called from the parent class of **Student**, that is, **Person**.
- **alice.graduates()** is called directly from the **Student** class itself.

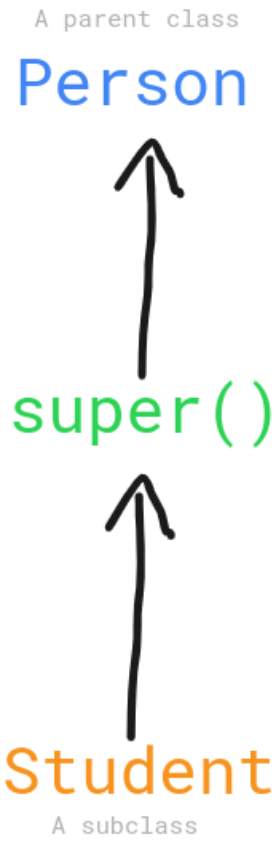
This demonstrates well how inheritance works.

Each **Student** is a **Person**. It makes sense to give the properties of a **Person** directly to the **Student**, instead of re-implementing them in the **Student** class.

Now you understand what inheritance is in Python.

Next, let's take a look at how the **super()** method works.

The super() in Python



To access parent class properties from the subclass, you need to use the **super()** method.

The **super()** is an explicit reference to the base class. It links your parent class to the child class.

Anything found in the parent class can be accessed in the child class via the **super()** method.

The most common example of using the **super()** method is upon initialization.

In the previous chapter, you already saw an example of this.

Now, let's take a deeper look at how it works.

The super().__init__() Call in Python

When you initialize a child class in Python, you can call the **super().__init__()** method. This initializes the parent class object into the child class.

In addition to this, you can add child-specific information to the child object as well.

Here is what it generally looks like:

```
class Parent:
    def init(v1, v2):
        self.v1 = v1
        self.v2 = v2

class Child(Parent):
    def init(v1, v2, v3):
        super().__init__(v1, v2)
        self.v3 = v3
```

Here:

- The **Parent** class has properties **v1** and **v2**. They are initialized in the parent classes **init()** method.
- The **Child** class inhertis the **Parent** class.
- The **Child** class initializes the **Parent** class object
- The **Child** class also initializes itself by specifying a new property **v3** that only belongs to it, not to the **Parent**.

Let's take a look at a more concrete example.

Let's say we have a class that represents a person.

Each person has a **name** and **age**.

Furthermore, each **Person** object has the ability to introduce themselves using the **introduce()** method.

Here is what the **Person** class looks like:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
```

```
        print(f"Hello, my name is {self.name}. I am {self.age} years old.")
```

This class acts as a blueprint for creating **Person** objects in our code.

Now, let's say we also want to represent students in our program.

To do this, we need a new class for student objects.

More specifically, each student should:

1. Have a name, age, and a graduation year.
2. Be able to introduce themselves.
3. Tell when they are going to graduate.

We could write a completely separate class like this:

```
class Student(Person):
    def __init__(self, name, age, graduation_year):
        self.name = name
        self.age = age
        self.graduation_year = graduation_year

    def introduce(self):
        print(f"Hello, my name is {self.name}. I am {self.age} years old.")

    def graduates(self):
        print(f"{self.name} will graduate in {self.graduation_year}")
```

Even though this works, there is a problem.

This code is now repetitive.

The **introduce()** method was already implemented in the **Person** class. Also, the **init()** method looks pretty similar.

We can improve the code by using inheritance.

The first thing to notice is that each **Student** is also a **Person**, which makes sense.

Thus we can inherit the properties of a **Person** to the **Student** class directly.

Let's then add a new member, **graduation_year**, to the **Student** object. In addition, we need a method to display this.

So when it comes to initializing the **Student** object, we can:

- Initialize the **Person** object in the **Student**. This happens with the **super().__init__()** call. This gives the **name** and **age** to the **Student** object.
- Initialize the **Student**-specific graduation year.

Here is the improved version of the **Student** class that utilizes inheritance.

```
class Student(Person):
    def __init__(self, name, age, graduation_year):
        # 1. Initialize the Person object in Student.
        super().__init__(name, age)
        # 2. Initialize the graduation_year
        self.graduation_year = graduation_year

    # Add a method that tells when this Student is going to graduate.
    def graduates(self):
        print(f"{self.name} will graduate in {self.graduation_year}")
```

Pay close attention to the **super().__init__(name, age)** call.

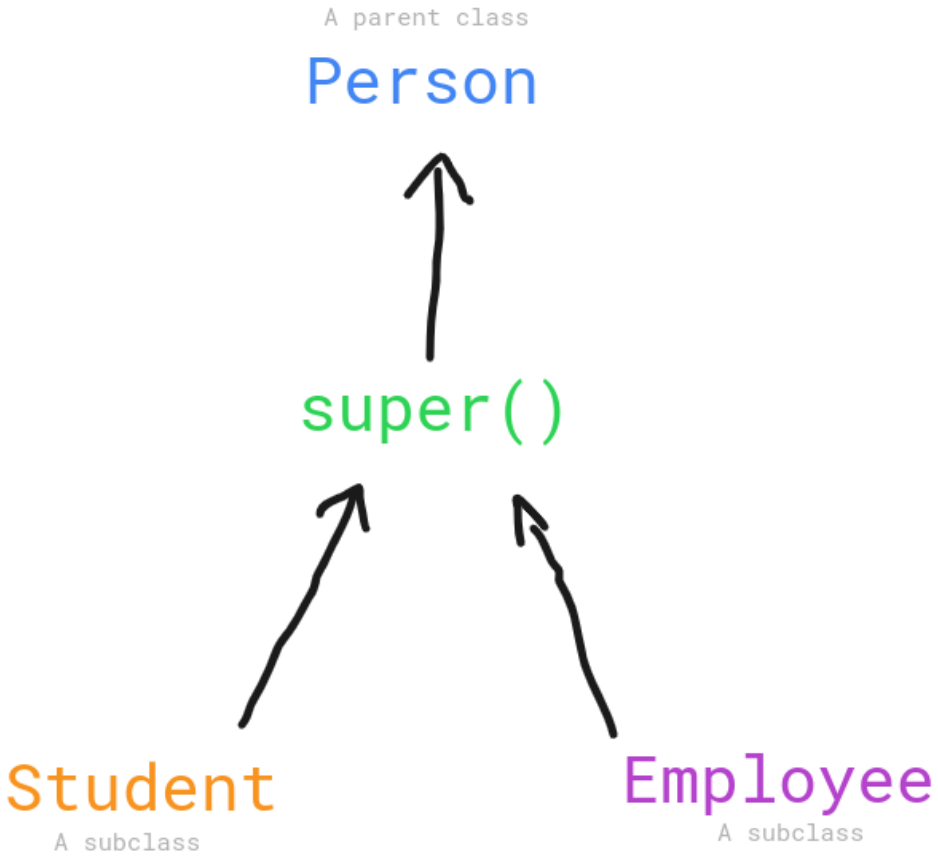
This calls the **__init__()** method of the parent class, **Person**.

In other words, it initializes a **Person** object into the **Student** object.

The super() Method in Multiple Inheritance

You can also streamline the process of initializing multiple classes with the help of the **super()** method.

In other words, you can use the **super()** method in multiple subclasses to access the common parent classes properties.



For instance, let's create a hierarchy such that a **Person** object is inherited to **Student** and **Employee**.

Here is how it looks in code:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"Hello, my name is {self.name}. I am {self.age} years old.")

# Subclass 1.
class Student(Person):
    def __init__(self, name, age, graduation_year):
        super().__init__(name, age)
        self.graduation_year = graduation_year

    def graduates(self):
        print(f"{self.name} will graduate in {self.graduation_year}")

# Subclass 2.
```

```
class Employee(Person):
    def __init__(self, name, age, start_year):
        super().__init__(name, age)
        self.start_year = start_year

    def graduates(self):
        print(f"{self.name} started working in {self.start_year}")
```

Access Regular Inherited Methods with Super()

In a couple of last examples, you saw how to use the **super()** method to call the initializer of the parent class.

It is important to notice you can access any other method too.

For example, let's modify the Person-Student example a bit. Let's create an **info()** method for the **Student** class. This method:

- Calls the **introduce()** method from the parent class to introduce itself.
- Shows the graduation year.

To call the **introduce()** method from the parent class, use the **super()** method to access it.

Here is how it looks in code:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"Hello, my name is {self.name}. I am {self.age} years old.")

# Subclass 1.
class Student(Person):
    def __init__(self, name, age, graduation_year):
        super().__init__(name, age)
        self.graduation_year = graduation_year

    def info(self):
```

```
        # Call the Person classes introduce() method to introduce this Stude
        super().introduce()
        print(f"{self.name} will graduate in {self.graduation_year}")

alice = Student("Alice", 30, 2023)
alice.info()
```

Output:

```
Hello, my name is Alice. I am 30 years old.
Alice will graduate in 2023
```

As you can see, now it is possible to call the **info()** method on a **Student** object to see the introduction and the graduation year prints.

So the **super()** method can be useful in many ways in Python.

This completes our guide.

Conclusion

Today you learned what the **super()** method does in Python.

To recap, the **super()** method links a parent class to its child class.

You can access all the properties in the parent class via the **super()** method.

Thanks for reading.

Happy coding!

Further Reading

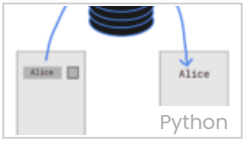
50 Python Interview Questions

About the Author

Artturi Jalli

I'm an entrepreneur and a blogger from Finland. My goal is to make coding and tech easier for you with comprehensive guides and reviews.

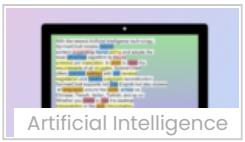
Recent Posts



2023.02.16
How to Pass a Variable from HTML Page to Another (w/ JavaScript)

2023.02.16
JavaScript How to Get Selected Text from a Textbox

2023.02.15
JavaScript Select All Text in Div with a Mouse Click



2023.02.13
Best Paraphrasing Tool | 10+ AI Rephrasing Tools Compared

Become a Job-Ready Developer

Get tips on how to become a job-ready software developer in no time.

Email Address

Subscribe

← Previous Post

Next Post →

Become a Job-Ready Developer

Get tips on how to become a job-ready software developer in no time.

Email Address

Subscribe

About the Author

Hi, I'm Artturi Jalli! ✌️

I'm a Tech enthusiast from Finland. 🇫🇮

👨‍🏫 I make Coding & Tech easy and fun with well-thought how-to guides and reviews.

🌍 I've already helped **3M+ visitors** reach their goals!

Contact

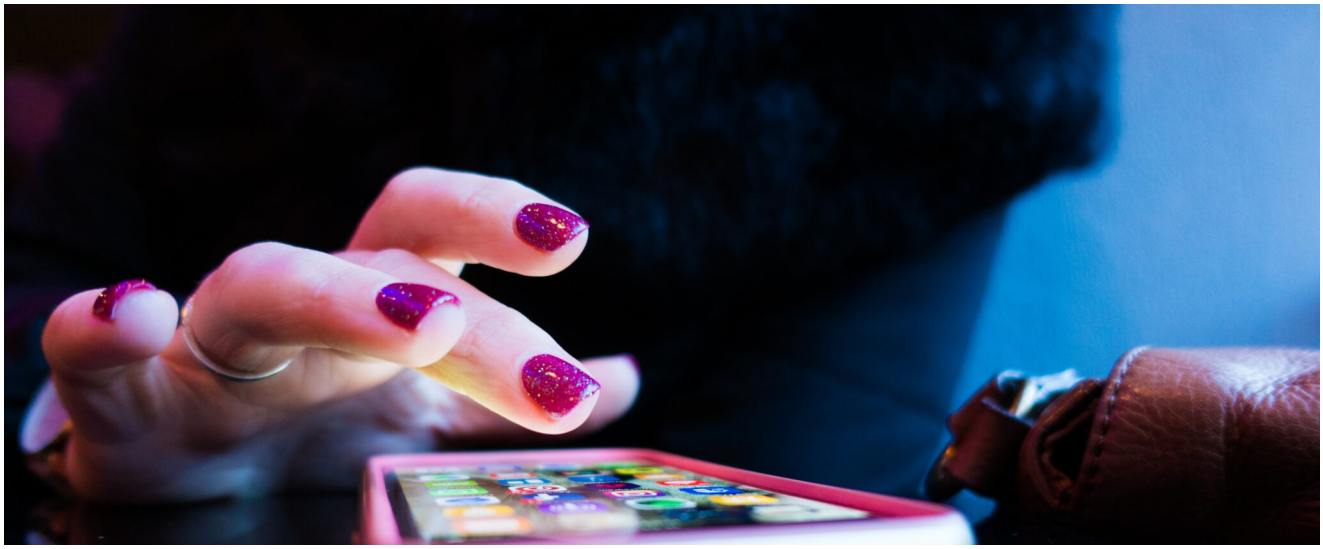
Search ...

ChatGPT—A Complete Guide to Using AI in Writing (2023)

ChatGPT is the newest Artificial Intelligence language model developed by OpenAI. Essentially, ChatGPT is an AI-based chatbot that can answer any question. It understands complex topics, like...

10+ Best AI Art Generators of February 2023 (I've Used Them All)

How to Make an App — A Complete 10-Step Guide (in 2023)



9 Best Graphic Design Courses + Certification (in 2023)

Do you want to become a versatile and skilled graphic designer? This is a comprehensive article on the best graphic design certification courses. These courses prepare you...

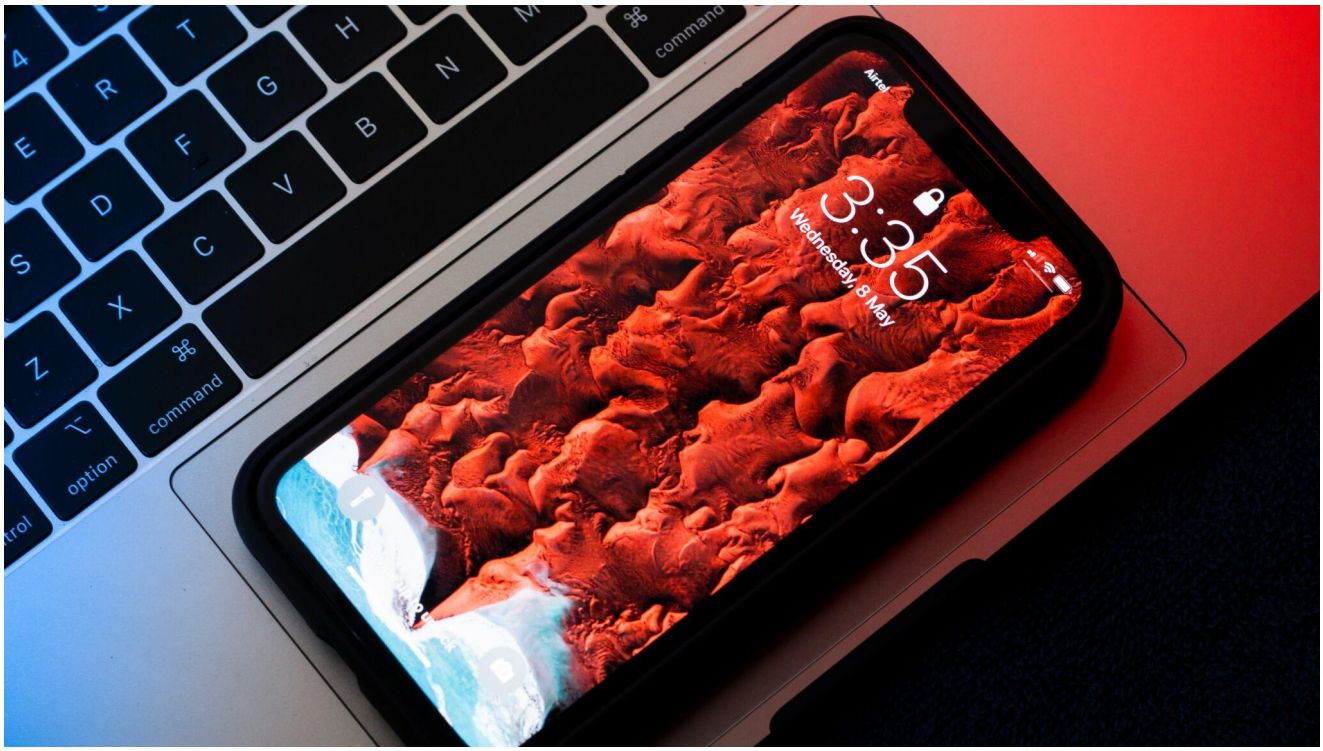
Continue Reading

8 Best Python Courses with Certifications (in 2023)

Are you looking to become a professional Python developer? Or are you interested in programming but don't know where to start? Python is a beginner-friendly and versatile...

Continue Reading

8 Best Swift & iOS App Development Courses [in 2023]



Are you looking to become an iOS developer? Do you want to create apps with an outstanding design? Do you want to learn to code? IOS App...

Continue Reading

Recent Posts

How to Pass a Variable from HTML Page to Another (w/ JavaScript)

JavaScript How to Get Selected Text from a Textbox

JavaScript Select All Text in Div with a Mouse Click

Best Paraphrasing Tool | 10+ AI Rephrasing Tools Compared

JavaScript How to Calculate Age from Birthdate

Categories

Artificial Intelligence

About

Codingem is a one-stop solution for all your coding needs. Whether you're working on a programming course, your own project, or in a professional setting, our website is here to help you save time and find the answers you're looking for.

Contact

artturi@codingem.com
Otakaari 7, 4th Floor, Espoo, Finland