

Super Function

In this lesson, you'll get to know about the uses of the super function in Python.

WE'LL COVER THE FOLLOWING ^

- What is the `super()` Function?
- Use Cases of the `super()` Function
 - Accessing Parent Class Properties
 - Calling Parent Class Methods
 - Using with Initializers

What is the `super()` Function?

The use of `super()` comes into play when we implement inheritance. It is used in a *child class* to **refer** to the *parent class* without explicitly naming it. It makes the code more manageable, and there is no need to know the name of the parent class to access its attributes.

Note: Make sure to add parenthesis at the end to avoid a compilation error.

`super()`



`super`



Use Cases of the `super()` Function

The super function is used in *three* relevant contexts:

Accessing Parent Class Properties

Consider the fields named `fuelCap` defined inside a `Vehicle` class to keep track of the *fuel capacity* of a vehicle. Another class named as `Car` extends from this `Vehicle` class. We declare a **class property** inside the `Car` class with the same name, i.e., `fuelCap` but different value. Now, if we want to refer to the `fuelCap` field of the *parent class* inside the *child class*, we will then have to use the `super()` function.

Let's understand this using the code below:

```
class Vehicle: # defining the parent class
    fuelCap = 90

class Car(Vehicle): # defining the child class
    fuelCap = 50

    def display(self):
        # accessing fuelCap from the Vehicle class using super()
        print("Fuel cap from the Vehicle Class:", super().fuelCap)

        # accessing fuelCap from the Vehicle class using self
        print("Fuel cap from the Car Class:", self.fuelCap)

obj1 = Car() # creating a car object
obj1.display() # calling the Car class method display()
```



Calling Parent Class Methods

Just like the properties, `super()` is also used with the methods. Whenever a *parent class* and the **immediate child class** have any methods with the same name, we use `super()` to access the methods from the parent class inside the child class. Let's go through an example:

```
class Vehicle: # defining the parent class
    def display(self): # defining display method in the parent class
        print("I am from the Vehicle Class")

class Car(Vehicle): # defining the child class
    # defining display method in the parent class
    def display(self):
        super().display()
        print("I am from the Car Class")
```



```
obj1 = Car() # creating a car object
obj1.display() # calling the Car class method printOut()
```



Using with Initializers

Another essential use of the function `super()` is to call the *initializer* of the *parent class* from the inside of the *initializer* of the *child class*.

Note: It is **not** necessary that the call to `super()` in a method or an initializer is made in the first line of the method.

Below is an example of using `super()` in initializer inside the child class.

 first

 second

```
class ParentClass():
    def __init__(self, a, b):
        self.a = a
        self.b = b


class ChildClass(ParentClass):
    def __init__(self, a, b, c):
        super().__init__(a, b)
        self.c = c

obj = ChildClass(1, 2, 3)
print(obj.a)
print(obj.b)
print(obj.c)
```



As you can see in both the code tabs, swapping the order of **line 9** and **line 10** does not change the functionality of the code. This allows the user to manipulate parameters before passing them into the parent class method.

Now let's use the example in the [previous lesson](#) and use `super()` to refer to the parent class:

without_super

with_super




```
class Vehicle:
    def __init__(self, make, color, model):
        self.make = make
        self.color = color
        self.model = model

    def printDetails(self):
        print("Manufacturer:", self.make)
        print("Color:", self.color)
        print("Model:", self.model)

class Car(Vehicle):
    def __init__(self, make, color, model, doors):
        Vehicle.__init__(self, make, color, model)
        self.doors = doors

    def printCarDetails(self):
        self.printDetails()
        print("Name:", self.doors)

obj1 = Car("Suzuki", "Grey", "2015", 4)
obj1.printCarDetails()
```



As you can see in the above codes, **line 15** is interchangeable and produces the same output but using `super()` makes the code more manageable.

So this was pretty much all about the `super()` function. In the next lesson, we will discuss the different types of inheritance.