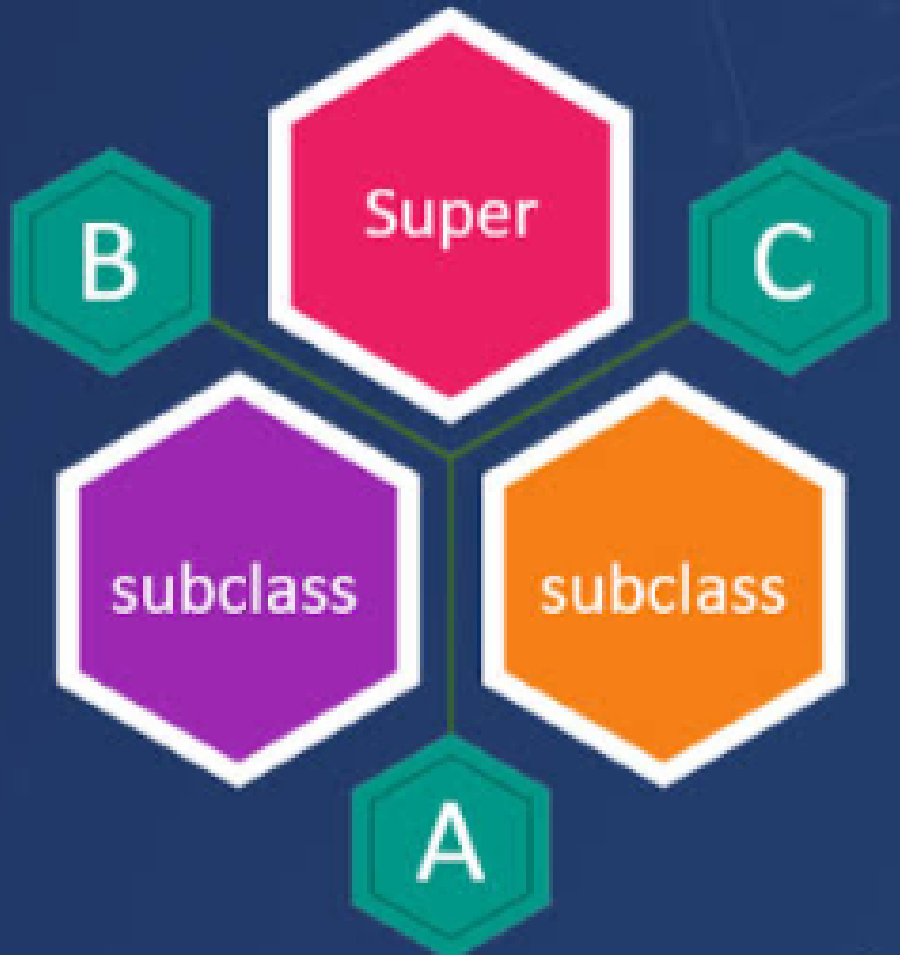
Python

super() in Python



super in python

Overview

The super function in Python is used to access methods of the immediate parent class.

Syntax of super() in Python

The syntax of super in python is given below:

Parameters of super() in Python

The super function in Python takes two optional parameters:

- ClassName: This is the name of the subclass.
- ClassObject: This is an object of the subclass.

Return Values of super() in Python

Return Type: <class 'super'>

The super in Python returns a temporary proxy object of the immediate parent class that can be used to call methods of the parent class.

Example of super() in Python

Let's understand the super in python using an example.

```
# Parent Class
class Vehicle:
  def __init__(self, company, model, year, color):
    self.company = company
    self.model = model
    self.year = year
    self.color = color

# Child Class
class Car(Vehicle):
  def __init__(self, company, model, year, color, car_type):
    # Using super to access __init__ method of Parent Class
    super().__init__(company, model, year, color)
    self.car_type = car_type

# Creating an object of Car
my_car = Car("Tesla", "S", 2021, "Silver", "Sedan")
print(f"I have a {my_car.company} model {my_car.model}.")
```

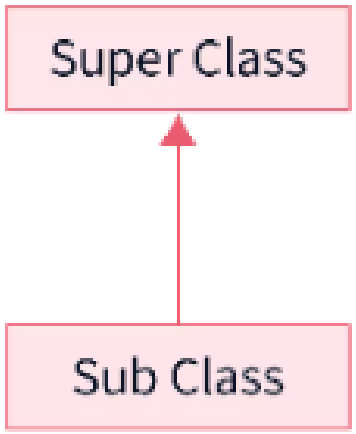
Output:

```
I have a Tesla model S.
```

What is super() Function in Python?

For understanding the super function, one should be familiar with the concept of classes and inheritance in Python.

Inheritance is a mechanism by which a class derives (or inherits) attributes and behaviors from another class without needing to implement them again.



Now, getting back to super. When do we use it?

Super is used when we need to build classes that extend the functionality of previously built classes. Let's understand this with an example.

Create a class Square:

```
class Square:
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side
```

Square has a side attribute and one method for calculating area.

Now, we will create a class Cube that inherits Square. With the help of super we can implement the surface_area method in Cube without having to re-write the method for calculating area:

```
class Cube(Square):
    def __init__(self, side):
      super().__init__(side)

    def surface_area(self):
        # Using super to access area()
        face_area = super().area()
        return face_area * 6

    def volume(self):
        face_area = super().area()
        return face_area * self.length
```

To summarise, super() returns a temporary object of the superclass that then allows us to call that superclass's methods.

Use of super() in Python

The benefits of using a super function are:

- No need to specify the parent class name.
- Helps in implementing modularity (isolating changes) and code reusability.

More Examples

Example 1: super() with Single Inheritance

Let's understand the use of the super() function in the single inheritance ie IS-A relationship. Till now, we have understood that the super() function is used to access the members of the immediate parent's class.

Let's understand how to call the parent's class method using the super() function in the single Inheritance.

In the below example, the dog class is inheriting the Animal class. This is an example of single-level inheritance. If we want to access the method of the animal class in the dog class, we can access them using the super() method.

Let's understand how to use super with single inheritance.

Let's understand how to use super with single inheritance.

```
class Animal:
    def smell(self,name):
        print(name, "can smell")

class Dog(Animal):
    def __init__(self):
        super().smell("Dog")

    def bark(self):
        print("Dog can bark")

dog = Dog()
dog.bark()
```

Output:

```
Dog can smell
Dog can bark
```

The super().smell() is calling the method of the parent's class method.

Example 2: super() with Multiple Inheritance

In multiple inheritance, a particular class can inherit as many classes.

Let's understand multiple inheritance using an example.

In the below example, the Child is inheriting two classes.

```
class Parent:
    def __init__(self):
        print("This is the parent class")

class Parent1:
    def __init__(self):
        print("This is the parent1 class")

class Child(Parent1, Parent):
    def __init__(self):
        ##Calling constructor of the Parnet 1 class
       super().__init__()

ob = Child()
```

Output:

```
This is the parent1 class
```

Now, how to access the constructor of the Parent class using the super() function? First of all, we have to understand the concept of the MRO.

MRO

MRO stands for Method Resolution Order. MRO defines the order of the inherited methods in the child class. Let's understand using the above example, In the above example we are accessing the constructor using the super() function, the super() will search the constructor according to the order of the inherited class, it will search first in the Parent1 class than in the Parent class.

In the above case, the constructor is already present in the Parent1 class that's why the constructor of the Parent1 class is executed instead of the Parent class, and also the super() function in python is used to access the immediate parent's class members.

Let's understand this using an example.

Let's understand the order of inheriting class by the Child class.

```
class Parent:
    def __init__(self):
        print("This is the parent class")

class Parent1:
    def __init__(self):
        print("This is the parent1 class")

class Child(Parent1, Parent):
    def __init__(self):
        ##Calling constructor of the Parnet 1 class
       super().__init__()

print(Child.mro())
```

Output:

```
[<class '__main__.Child'>, <class '__main__.Parent1'>, <class '__main__.Parent'>, <class
```

We can see clearly in the output, the Child class first extends the Parent1 class then extend the Parent class. The object class is the super class of all the classes in the python language that's why at last the Child class also extends the object class by default.

Now how to access the method of the Parent class using the super function? Let's understand the syntax of the super() function to access the Parent class methods using the super() function.

Syntax:

```
Super(Immediateclassname,currentobject)
```

This **super() function** accepts two parameters in the multiple inheritance.

- **ImmediateClassName**: The name of the class that is just inherited before the class that we want to access using the super() function.
- **currentObject**: The current object of the class.

Let's understand how to access the constructor of the Parent class from the Child class.

```
class Parent:
    def __init__(self):
        print("This is the parent class")

class Parent1:
    def __init__(self):
        print("This is the parent1 class")

class Child(Parent1, Parent):
    def __init__(self):
        ##Calling constructor of the Parent 1 class
       super(Parent1, self).__init__()

ob = Child()
```

Output:

```
This is the parent class
```

The constructor of the Parent class is accessed using the super() function in the Child class.

Conclusion

- The super function in Python is used to access methods of the immediate parent class.
- The Return type of the Super function is a proxy object of the Immediate class.
- The super() function use the concept of MRO in the multiple inheritance.
- The super() function also takes two parameters i.e the immediate parent class name and the current object.