

Importance of super () and how to call it in child class

What the point of using parent class if I commenting out the class parent class still the output is same

```
class Restaurant :
    """A simple restaurant class"""
    def init(self,restaurant_name,cuisine_type,number_served=0):
        self.restaurant_name =restaurant_name
        self.cuisine_type =cuisine_type
        self.number_served = number_served
    def describe_restaurant(self) :
        print(f"Name of the restaurant - {self.restaurant_name} and served {self.cuisine_type}")
        print(f"{self.restaurant_name} served {self.cuisine_type}")
    def open_restaurant(self) :
        print(f"{self.restaurant_name} is now OPEN ")
    def set_number_served(self,served) :
        self.number_served = served
    def increment_number_served(self,add_served) :
        self.number_served += add_served
class IceCreamStand (Restaurant) :
    def init(self,restaurant_name,cuisine_type,flavours,number_served):
        """Initialize attributes of the parentclass"""
        #super ().init(restaurant_name,cuisine_type,number_served=0)
        self.flavours = flavours
    def display_flavours(self):
        print ("Available flavours are : ")
        for flavour in self.flavours :
            print(f" - {flavour}")
all_flavours = IceCreamStand('Muse', 'Mexican',['orange','pinapple','mango'],90)

all_flavours.display_flavours()
my_restaurant= Restaurant('Muse', 'Mexican')
my_restaurant.describe_restaurant()
my_restaurant.open_restaurant()

my_restaurant.set_number_served (94)
print(f"Total {my_restaurant.number_served} number of great customer served ")
my_restaurant.increment_number_served (6)
print(f"And calling the day off with {my_restaurant.number_served} customers in total.")
```

Good question! You are right that commenting out the parent class (`Restaurant`) initialization in the `IceCreamStand` class does not immediately break the code or change the output in your case. But this is because the specific part of the parent class (`Restaurant`) you are commenting out is not directly used in the `IceCreamStand` class in this current setup.

Here's why the inheritance is still important, even though it seems to work without it in your specific example:

1. Inheritance Provides Reusability and Flexibility:

By inheriting from `Restaurant`, the `IceCreamStand` class can reuse all the methods and attributes of `Restaurant`. This means you don't have to rewrite methods like `describe_restaurant()`, `open_restaurant()`, `set_number_served()`, or `increment_number_served()`. Even though in this specific case, you're not using those methods in the `IceCreamStand` class, inheritance gives you the option to do so later without duplicating code.

2. Why Your Current Code Works Without It:

In your code, you haven't used any `Restaurant` class methods or attributes inside `IceCreamStand`. Since the only thing the `IceCreamStand` class is doing is handling flavors (via the `display_flavours()` method), you don't immediately notice the absence of `super()` because you're not using any of the inherited functionality.

If you commented out the parent class initialization (`super().__init__()`) but tried to access any `Restaurant` attributes (like `restaurant_name`, `cuisine_type`, or `number_served`) inside `IceCreamStand`, you would get an error because those attributes wouldn't be initialized.

Example Showing the Importance of `super()` :

Let's modify the `IceCreamStand` class so it uses the `Restaurant` attributes. You'll see the need for `super()` in this case.

```
class IceCreamStand(Restaurant):
    def __init__(self, restaurant_name, cuisine_type, flavours, number_served=0):
        """Initialize attributes of the parent class"""
        super().__init__(restaurant_name, cuisine_type, number_served) # Call parent init
        self.flavours = flavours

    def display_flavours(self):
        print(f"{self.restaurant_name} offers the following flavours:") # Using Restaurant
        attribute
        for flavour in self.flavours:
            print(f" - {flavour}")
```

Now, the `display_flavours()` method uses `self.restaurant_name`, which is defined in the parent class (`Restaurant`). If you comment out the `super().__init__()` line, the code will break because `restaurant_name` won't be initialized in `IceCreamStand`.

3. Code Duplication:

If you don't use inheritance, you might need to copy the same methods into multiple classes if they share functionality (like `open_restaurant()` or `describe_restaurant()`). This makes the code harder to maintain. If you need to change the behavior of a shared method later, you'd have to update it in multiple places instead of just the parent class.

Conclusion:

In your current code, inheritance might not seem necessary because `IceCreamStand` is not using any parent class methods or attributes. But in a real-world scenario, you'd likely want the `IceCreamStand` class to have the functionality of the `Restaurant` class (like setting the number of customers served or opening the restaurant), and

that's when `super()` and inheritance will become essential. It helps avoid code duplication and makes the code easier to maintain and extend.

how to access parent class attribute then where `super` is used

Great! Let's dive into how to properly access parent class attributes when using `super()` in the child class.

When you use `super()` in the child class, you are initializing the parent class (in this case, `Restaurant`) and gaining access to all its methods and attributes.

Here's an example of how you can access parent class attributes after calling `super()` in the child class:

Complete Example

```
# Parent Class: Restaurant
class Restaurant:
    """A simple restaurant class."""
    def __init__(self, restaurant_name, cuisine_type, number_served=0):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type
        self.number_served = number_served

    def describe_restaurant(self):
        print(f"Restaurant: {self.restaurant_name} serves {self.cuisine_type} cuisine.")

    def open_restaurant(self):
        print(f"{self.restaurant_name} is now OPEN!")

    def set_number_served(self, served):
        self.number_served = served

    def increment_number_served(self, add_served):
        self.number_served += add_served

# Child Class: IceCreamStand (inherits from Restaurant)
class IceCreamStand(Restaurant):
    """A specialized version of a restaurant that serves ice cream."""
    def __init__(self, restaurant_name, cuisine_type, flavours, number_served=0):
        # Use super() to initialize attributes from the parent class (Restaurant)
        super().__init__(restaurant_name, cuisine_type, number_served)
        self.flavours = flavours

    # Method specific to IceCreamStand to display the ice cream flavours
    def display_flavours(self):
        print(f"{self.restaurant_name} offers the following ice cream flavours:")
        for flavour in self.flavours:
            print(f" - {flavour}")

# Creating an instance of IceCreamStand and accessing parent class attributes
my_ice_cream_stand = IceCreamStand('Chillz Delight', 'Desserts', ['Vanilla', 'Chocolate', 'Strawberry'], 50)

# Accessing methods from the parent class
my_ice_cream_stand.describe_restaurant() # This method is from the Restaurant class
my_ice_cream_stand.open_restaurant()    # This method is from the Restaurant class

# Accessing the number of customers served, an attribute inherited from Restaurant
print(f"Number of customers served: {my_ice_cream_stand.number_served}")
```

```
# Using the IceCreamStand-specific method to display flavours
my_ice_cream_stand.display_flavours()
```

Output

```
Restaurant: Chillz Delight serves Desserts cuisine.
Chillz Delight is now OPEN!
Number of customers served: 50
Chillz Delight offers the following ice cream flavours:
- Vanilla
- Chocolate
- Strawberry
```

Key Takeaways:

- `super().__init__()` is used in the `IceCreamStand` class to initialize the attributes from the `Restaurant` parent class (`restaurant_name` , `cuisine_type` , and `number_served`).
- After calling `super()` , the `IceCreamStand` object (`my_ice_cream_stand`) can access both the parent class methods (`describe_restaurant()` , `open_restaurant()`) and its attributes (`restaurant_name` , `number_served`).
- The `display_flavours()` method is unique to `IceCreamStand` , but it still uses the `restaurant_name` attribute inherited from `Restaurant` .

This demonstrates how you can effectively use `super()` to inherit attributes and methods from a parent class, and then extend or specialize them in the child class.