### Base Class (Superclass):

**Definition:** The class whose attributes and methods are inherited by another class.

**Alternative Names:** Superclass, Parent Class.

**Role:** Provides a general template or blueprint for creating objects.

**Example:**

class Animal:

def eat(self):

print("Animal is eating")

### Derived Class (Subclass):

**Definition:** The class that inherits attributes and methods from another class.

**Alternative Names:** Subclass, Child Class.

**Role:** Specializes or extends the functionality of the base class.

**Example:**

class Dog(Animal):

def bark(self):

print("Dog is barking")

### Super() Function:

**Definition:** A built-in function used to call a method from the parent class within the subclass.

**Purpose:** Invokes the method of the base class, allowing the subclass to extend rather than replace the functionality.

**Example:**

class Animal:

def make\_sound(self):

print("Generic animal sound")

class Dog(Animal):

def make\_sound(self):

super().make\_sound()

print("Woof!")

These simplified concepts still capture the essence of inheritance in object-oriented programming. If you have any specific questions or if there's anything else you'd like to explore, feel free to let me know!

Certainly! The super() function in Python is used to call a method from a parent or superclass. It's often used within a subclass to invoke a method or constructor of its immediate parent class. This allows you to extend or override the functionality of the parent class while still utilizing its methods.

### Using super() in Methods:

Let's say you have a base class Animal with a method speak():

class Animal:

def speak(self):

print("Generic animal sound")

Now, you want to create a subclass Dog that extends the Animal class and provides its own implementation of the speak() method:

class Dog(Animal):

def speak(self):

print("Woof!")

However, if you still want to use the functionality of the speak() method from the Animal class within the Dog class, you can use super():

class Dog(Animal):

def speak(self):

super().speak() # Calls the speak method from the Animal class

print("Woof!")

In this example, super().speak() calls the speak() method of the parent class (Animal). This is useful when you want to extend the functionality of the parent class method rather than completely replacing it.

### Using super() in Constructors:

Similarly, super() can be used in constructors to invoke the constructor of the parent class:

class Animal:

def \_\_init\_\_(self, species):

self.species = species

class Dog(Animal):

def \_\_init\_\_(self, species, breed):

super().\_\_init\_\_(species) # Calls the constructor of the Animal class

self.breed = breed

In this case, super().\_\_init\_\_(species) ensures that the species attribute is initialized using the constructor of the Animal class before adding the breed attribute in the Dog class.

Using super() helps maintain a clear hierarchy and ensures that the code is more robust when dealing with multiple levels of inheritance.

# Purpose

Certainly! Inheritance in programming is like passing on traits from one class to another. Imagine you have a basic class called "Animal" with common features like eating.

Now, if you want to create a more specific class, like "Dog," you can use inheritance. The "Dog" class automatically gets the ability to eat (inherited from the "Animal" class). This way, you avoid rewriting the same code for eating in both "Animal" and "Dog."

So, the main purposes of inheritance are:

1. **Reuse Code:**
   * Inherit common features from one class to another, avoiding repetition.
2. **Organization:**
   * Create a clear structure where classes build upon each other, making your code easy to understand.
3. **Customization:**
   * Customize or add new features in specific classes without affecting others.

Think of it like creating a family tree for your code, where each class inherits traits from its ancestors. It makes your code more flexible, readable, and easier to manage.

In this example, we define a GFG class with one instance variable name representing the name of a person. The **str()** method is defined to return a human-readable string representation of the instance, while the **repr()** method is defined to return a string representation of the instance that can be used to recreate the object.