**Inheritance in Python**

**Inheritance** is a fundamental concept in object-oriented programming (OOP) that allows a class (known as a **child class** or **subclass**) to inherit attributes and methods from another class (known as a **parent class** or **superclass**). Inheritance promotes code reuse and creates a logical hierarchy between classes.

**Types of Inheritance in Python**

1. **Single Inheritance**
2. **Multiple Inheritance**
3. **Multilevel Inheritance**
4. **Hierarchical Inheritance**
5. **Hybrid Inheritance**

Let’s go over each type with detailed explanations and examples.

**1. Single Inheritance**

In **single inheritance**, a subclass inherits from a single superclass. This is the simplest form of inheritance.

**Example:**

python

Copy code

# Parent class (superclass)

class Animal:

def speak(self):

return "Animal makes a sound"

# Child class (subclass)

class Dog(Animal):

def speak(self):

return "Dog barks"

# Creating an instance of Dog

dog = Dog()

print(dog.speak()) # Output: Dog barks

**Explanation:**

* **Animal** is the parent class with a speak() method.
* **Dog** inherits from Animal and overrides the speak() method to provide a dog-specific implementation.
* This is a basic example of single inheritance where one class (Dog) inherits from one parent class (Animal).

**2. Multiple Inheritance**

In **multiple inheritance**, a subclass can inherit from more than one parent class. This can sometimes lead to complexity (such as the "Diamond Problem"), but Python handles it using the **Method Resolution Order (MRO)**.

**Example:**

python

Copy code

# Parent class 1

class Father:

def work(self):

return "Father is working"

# Parent class 2

class Mother:

def cook(self):

return "Mother is cooking"

# Child class inheriting from both Father and Mother

class Child(Father, Mother):

def play(self):

return "Child is playing"

# Creating an instance of Child

child = Child()

print(child.work()) # Output: Father is working

print(child.cook()) # Output: Mother is cooking

print(child.play()) # Output: Child is playing

**Explanation:**

* **Child** inherits from both Father and Mother, allowing the Child to access methods from both parent classes.
* The child class can inherit methods from both parents and also define its own methods (e.g., play()).

**3. Multilevel Inheritance**

In **multilevel inheritance**, a subclass inherits from a class, which in turn inherits from another class, forming a chain of inheritance.

**Example:**

python

Copy code

# Base class (superclass)

class Animal:

def sound(self):

return "Animal makes a sound"

# Intermediate class (subclass of Animal)

class Mammal(Animal):

def walk(self):

return "Mammal walks"

# Child class (subclass of Mammal)

class Dog(Mammal):

def bark(self):

return "Dog barks"

# Creating an instance of Dog

dog = Dog()

print(dog.sound()) # Output: Animal makes a sound

print(dog.walk()) # Output: Mammal walks

print(dog.bark()) # Output: Dog barks

**Explanation:**

* The **Dog** class inherits from **Mammal**, and **Mammal** inherits from **Animal**.
* This creates a hierarchy of inheritance where the child class (Dog) inherits methods from both the base class (Animal) and intermediate class (Mammal).

**4. Hierarchical Inheritance**

In **hierarchical inheritance**, multiple subclasses inherit from the same parent class. Each subclass may have its own additional methods.

**Example:**

python

Copy code

# Base class (superclass)

class Animal:

def sound(self):

return "Animal makes a sound"

# Child class 1

class Dog(Animal):

def bark(self):

return "Dog barks"

# Child class 2

class Cat(Animal):

def meow(self):

return "Cat meows"

# Creating instances of Dog and Cat

dog = Dog()

cat = Cat()

print(dog.sound()) # Output: Animal makes a sound

print(dog.bark()) # Output: Dog barks

print(cat.sound()) # Output: Animal makes a sound

print(cat.meow()) # Output: Cat meows

**Explanation:**

* Both **Dog** and **Cat** inherit from the **Animal** class.
* This allows each child class to inherit common functionality (like sound()), but also define its specific behavior (like bark() and meow()).

**5. Hybrid Inheritance**

**Hybrid inheritance** is a combination of two or more types of inheritance. It may involve a mixture of single, multiple, and multilevel inheritance structures.

**Example:**

python

Copy code

# Base class

class Animal:

def sound(self):

return "Animal makes a sound"

# Intermediate class 1

class Mammal(Animal):

def walk(self):

return "Mammal walks"

# Intermediate class 2

class Bird(Animal):

def fly(self):

return "Bird flies"

# Child class (Multiple inheritance)

class Bat(Mammal, Bird):

def echo(self):

return "Bat uses echolocation"

# Creating an instance of Bat

bat = Bat()

print(bat.sound()) # Output: Animal makes a sound (inherited from Animal)

print(bat.walk()) # Output: Mammal walks (inherited from Mammal)

print(bat.fly()) # Output: Bird flies (inherited from Bird)

print(bat.echo()) # Output: Bat uses echolocation

**Explanation:**

* **Bat** inherits from both **Mammal** and **Bird**, which in turn inherit from **Animal**.
* This is an example of **hybrid inheritance**, combining both multiple and multilevel inheritance.

**Method Resolution Order (MRO)**

In Python, when a class inherits from multiple parent classes, the **Method Resolution Order (MRO)** determines the order in which the base classes are searched for a method. You can check the MRO using the \_\_mro\_\_ attribute or the mro() method.

**Example:**

python

Copy code

print(Bat.mro())

**Output:**

arduino

Copy code

[<class '\_\_main\_\_.Bat'>, <class '\_\_main\_\_.Mammal'>, <class '\_\_main\_\_.Bird'>, <class '\_\_main\_\_.Animal'>, <class 'object'>]

This means that when you call a method on an instance of the Bat class, Python will first look in the Bat class, then in Mammal, followed by Bird, then Animal, and finally in the object class (the root of all classes in Python).

**Conclusion**

1. **Single Inheritance**: Inherits from one parent class.
2. **Multiple Inheritance**: Inherits from more than one parent class.
3. **Multilevel Inheritance**: Inherits through a chain of classes.
4. **Hierarchical Inheritance**: Multiple subclasses inherit from the same parent class.
5. **Hybrid Inheritance**: Combines two or more types of inheritance.

Inheritance helps in code reuse, logical classification, and improves the maintainability of the code by allowing the child classes to inherit properties and behavior from parent classes.