**AI Day 14 Notes**

**Syed Mansoor ul Hassan Bukhari**

**Inheritance in Python:**

1. **Definition**: Inheritance allows a class to inherit attributes and methods from another class. The class that inherits is called the child class, and the class being inherited from is called the parent class.

class Parent:

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

self.name = name

class Child(Parent):

def \_\_init\_\_(self, name, age):

super().\_\_init\_\_(name)

self.age = age

child = Child("John", 12)

print(child.name, child.age)

1. **Parent Class**: The parent class is the class being inherited from, also called the base class.

class Parent:

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

self.name = name

parent = Parent("John")

print(parent.name)

1. **Child Class**: The child class is the class that inherits from another class, also called the derived class.

class Parent:

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

self.name = name

class Child(Parent):

def \_\_init\_\_(self, name, age):

super().\_\_init\_\_(name)

self.age = age

child = Child("John", 12)

print(child.name, child.age)

1. **super() Function**: The super() function allows us to call methods of the parent class in the child class.

class Parent:

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

self.name = name

class Child(Parent):

def \_\_init\_\_(self, name, age):

super().\_\_init\_\_(name)

self.age = age

child = Child("John", 12)

print(child.name, child.age)

1. **Method Overriding**: Child classes can override methods from the parent class.

class Parent:

def show(self):

print("Parent method")

class Child(Parent):

def show(self):

print("Child method")

child = Child()

child.show()

1. **Multiple Inheritance**: A class can inherit from multiple classes.

class Parent1:

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

self.name = name

class Parent2:

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

self.age = age

class Child(Parent1, Parent2):

def \_\_init\_\_(self, name, age):

Parent1.\_\_init\_\_(self, name)

Parent2.\_\_init\_\_(self, age)

child = Child("John", 12)

print(child.name, child.age)

1. **Multilevel Inheritance**: A class can inherit from another class, which in turn inherits from another class.

class Grandparent:

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

self.name = name

class Parent(Grandparent):

def \_\_init\_\_(self, name, age):

super().\_\_init\_\_(name)

self.age = age

class Child(Parent):

def \_\_init\_\_(self, name, age, grade):

super().\_\_init\_\_(name, age)

self.grade = grade

child = Child("John", 12, "7th")

print(child.name, child.age, child.grade)

1. **Hierarchical Inheritance**: Multiple classes inherit from the same parent class.

class Parent:

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

self.name = name

class Child1(Parent):

def \_\_init\_\_(self, name, age):

super().\_\_init\_\_(name)

self.age = age

class Child2(Parent):

def \_\_init\_\_(self, name, grade):

super().\_\_init\_\_(name)

self.grade = grade

child1 = Child1("John", 12)

child2 = Child2("Doe", "7th")

print(child1.name, child1.age)

print(child2.name, child2.grade)

1. **Hybrid Inheritance**: A combination of two or more types of inheritance.

class Parent:

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

self.name = name

class Child1(Parent):

def \_\_init\_\_(self, name, age):

super().\_\_init\_\_(name)

self.age = age

class Child2(Parent):

def \_\_init\_\_(self, name, grade):

super().\_\_init\_\_(name)

self.grade = grade

class Grandchild(Child1, Child2):

def \_\_init\_\_(self, name, age, grade):

Child1.\_\_init\_\_(self, name, age)

Child2.\_\_init\_\_(self, name, grade)

grandchild = Grandchild("John", 12, "7th")

print(grandchild.name, grandchild.age, grandchild.grade)

1. **Accessing Parent Class Attributes**: Attributes of the parent class can be accessed using the super() function.

class Parent:

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

self.name = name

class Child(Parent):

def \_\_init\_\_(self, name, age):

super().\_\_init\_\_(name)

self.age = age

child = Child("John", 12)

print(child.name, child.age)

1. **Accessing Parent Class Methods**: Methods of the parent class can be accessed using the super() function.

class Parent:

def show(self):

print("Parent method")

class Child(Parent):

def show(self):

super().show()

print("Child method")

child = Child()

child.show()

1. **Using isinstance()**: The isinstance() function checks if an object is an instance of a class or a subclass thereof.

class Parent:

pass

class Child(Parent):

pass

child = Child()

print(isinstance(child, Child)) # True

print(isinstance(child, Parent)) # True

1. **Using issubclass()**: The issubclass() function checks if a class is a subclass of another class.

class Parent:

pass

class Child(Parent):

pass

print(issubclass(Child, Parent)) # True

print(issubclass(Parent, Child)) # False

1. **Private Members**: Private members of the parent class are not accessible directly in the child class.

class Parent:

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

self.\_\_name = name

class Child(Parent):

def \_\_init\_\_(name, age):

super().\_\_init\_\_(name)

self.age = age

child = Child("John", 12)

# print(child.\_\_name) # AttributeError

1. **Protected Members**: Protected members of the parent class can be accessed in the child class.

class Parent:

def \_\_init\_\_(name):

self.\_name = name

class Child(Parent):

def \_\_init\_\_(name, age):

super().\_\_init\_\_(name)

self.age = age

child = Child("John", 12)

print(child.\_name)

1. **Method Resolution Order (MRO)**: The order in which methods are resolved in the presence of multiple inheritance.

class A:

def show(self):

print("A method")

class B(A):

def show(self):

print("B method")

class C(A):

def show(self):

print("C method")

class D(B, C):

pass

d = D()

d.show() # B method

print(D.mro()) # [D, B, C, A, object]

1. **Diamond Problem**: The diamond problem occurs when a class inherits from two classes that have a common base class.

class A:

def show(self):

print("A method")

class B(A):

def show(self):

print("B method")

class C(A):

def show(self):

print("C method")

class D(B, C):

pass

d = D()

d.show() # B method

print(D.mro()) # [D, B, C, A, object]