Tops Technology

Module 15) Advance Python Programming

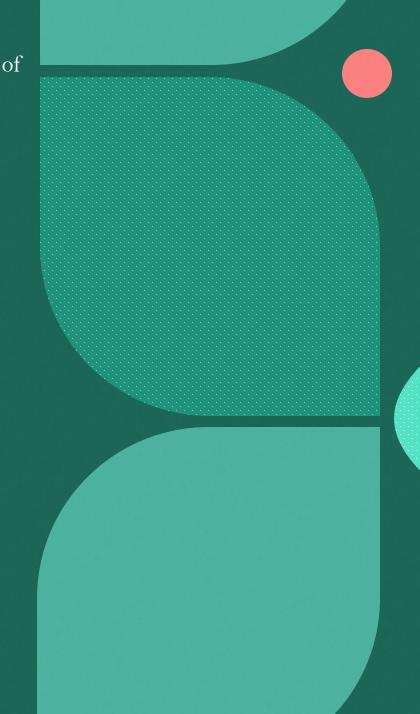
Presented By:

Nandni Vala

Inheritance

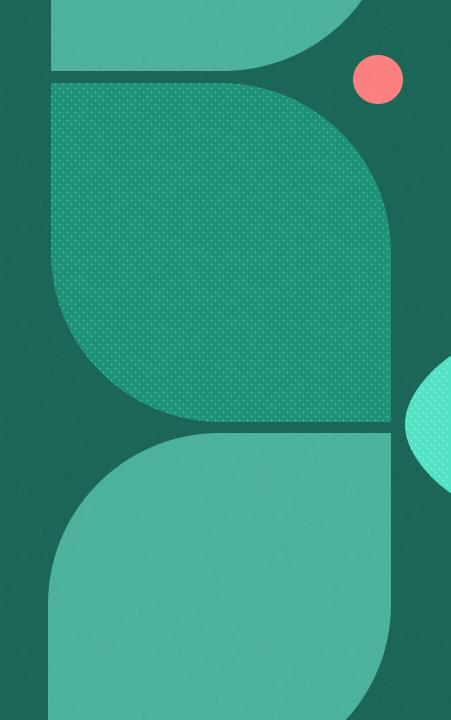
- 1. Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python.
- Inheritance is an object-oriented programming (OOP) concept where a class (child or derived) can inherit attributes and methods from another class (parent or base). Python supports different types of inheritance.
- > Single Inheritance
- ➤ A child class inherits from a single parent class.
- > Example:
- > class Parent:
- def display(self):
- print("This is the parent class.")
- > class Child(Parent):
- def show(self):
- print("This is the child class.")
- \triangleright obj = Child()
- ➤ obj.display() # Inherited method
- ➤ obj.show() # Child's own method

- > Multilevel Inheritance
- A class inherits from another class, and another class inherits from it (a chain of inheritance).
- **Example:**
- > class Grandparent:
- def show_grandparent(self):
- print("This is the grandparent class.")
- class Parent(Grandparent):
- def show_parent(self):
- print("This is the parent class.")
- > class Child(Parent):
- ➤ def show_child(self):
- print("This is the child class.")
- \triangleright obj = Child()
- bobj.show_grandparent() # Inherited from Grandparent
- b obj.show_parent() # Inherited from Parent
- ➤ obj.show_child() # Child's own method



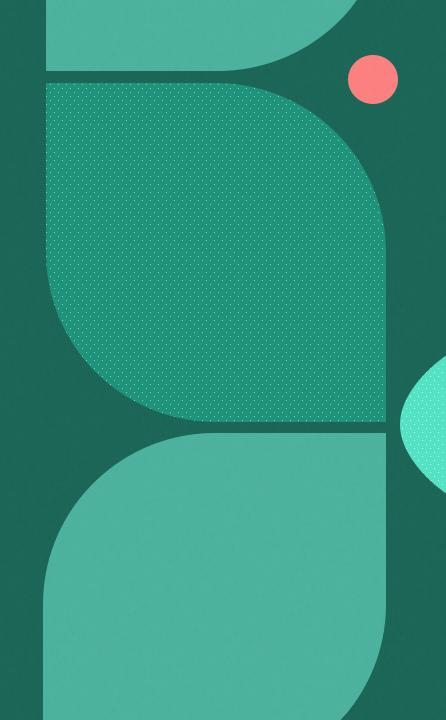
Multiple Inheritance

• A class inherits from more than one parent class. Example: class Parent 1: def display1(self): print("This is Parent 1.") class Parent2: def display2(self): print("This is Parent2.") class Child(Parent 1, Parent 2): def display_child(self): print("This is the child class.") obj = Child()# From Parent1 obj.display1() obj.display2() # From Parent2 obj.display_child() # Child's own method



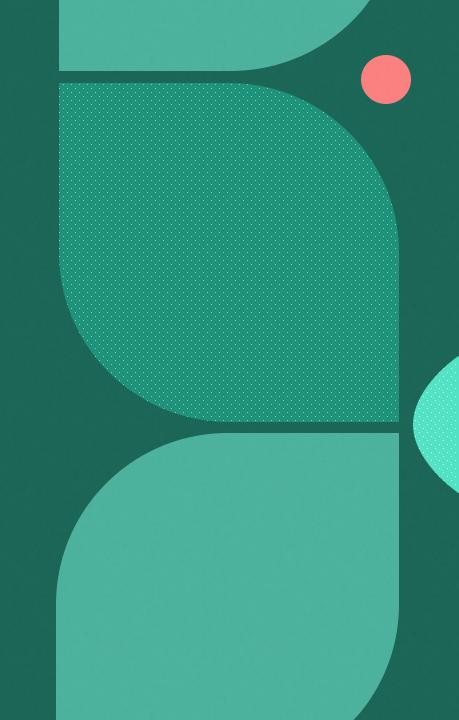
➤ Hierarchical Inheritance

- ➤ Multiple child classes inherit from the same parent class.
- **Example:**
- > class Parent:
- def show_parent(self):
- print("This is the parent class.")
- ➤ class Child1(Parent):
- def show_child1(self):
- print("This is Child1.")
- ➤ class Child2(Parent):
- def show_child2(self):
- print("This is Child2.")
- \triangleright obj1 = Child1()
- ➤ obj1.show_parent()
- ➤ obj1.show_child1()
- \triangleright obj2 = Child2()
- ➤ obj2.show_parent()
- ➤ obj2.show_child2()

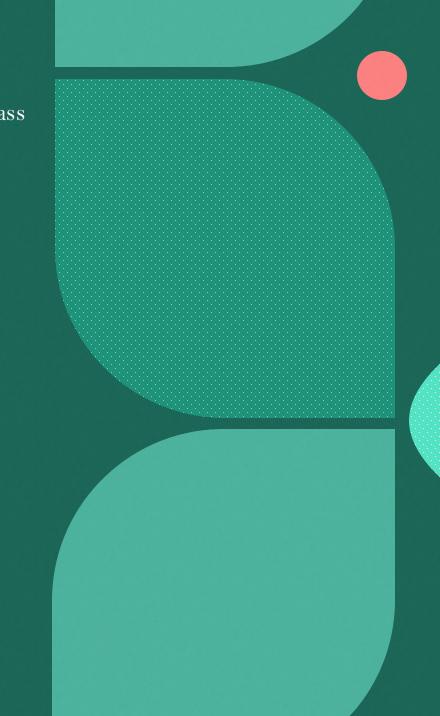


Hybrid Inheritance

- A combination of two or more types of inheritance.
- Example:
- class Base:
- def show_base(self):
- print("This is the base class.")
- class Child 1 (Base): # Single Inheritance
- def show_child1(self):
- print("This is Child 1.")
- class Child2(Base): # Hierarchical Inheritance
- def show_child2(self):
- print("This is Child2.")
- class Grandchild(Child1, Child2): # Multiple Inheritance
- def show_grandchild(self):
- print("This is the grandchild class.")
- obj = Grandchild()
- obj.show_base() # Inherited from Base
- obj.show_child1() # Inherited from Child1
- obj.show_child2() # Inherited from Child2
- obj.show_grandchild() # Grandchild's own method



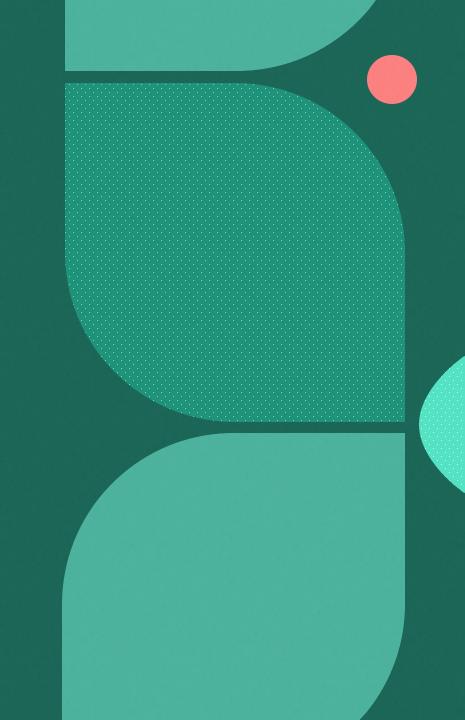
- 2. Using the super() function to access properties of the parent class.
- > super() Function in Python
- The super() function is used to call methods or access properties of a parent class from the child class. It allows you to:
- > Reuse the functionality of the parent class without explicitly naming it.
- Make code maintainable by avoiding hardcoding the parent class name.
- > Syntax
- super().method_name(args)
- > Key Use Cases of super()
- > Accessing Parent Class Methods
- Accessing Parent Class Constructor (__init__)
- > Avoiding Redundancy in Multiple Inheritance
- > Accessing Parent Class Methods
- ➤ Using super() to call a method in the parent class.





> Accessing Parent Class Constructor

- ➤ Using super() to invoke the parent class's __init__ method.
- > Example:
- > class Parent:
- def __init__(self, name):
- > self.name = name
- print(f"Parent initialized with name: {self.name}")
- ➤ class Child(Parent):
- def __init__(self, name, age):
- > super().__init__(name) # Call parent class constructor
- > self.age = age
- print(f"Child initialized with age: {self.age}")
- \triangleright obj = Child("John", 12)
- ➤ Output:
- Parent initialized with name: John
- ➤ Child initialized with age: 12



Using super() in Multiple Inheritance

In multiple inheritance, super() ensures that the **Method Resolution Order** (MRO) is followed.

```
Example:
class A:
  def display(self):
     print("Class A")
class B(A):
  def display(self):
     super().display()
     print("Class B")
class C(B):
  def display(self):
     super().display(
     print("Class C")
obj = C()
obj.display()
```

