PRACTICAL – 3

=>INHERITANCE<=

PROGRAM -1

AIM- WAPP FOR SINGLE INHERITANCE

CODE-

|  |
| --- |
| """  This program demonstrates inheritance in Python  """  print("HARSH D")  # Define the parent class  class parent:  def fun1(self):  """  This method prints a message indicating that it belongs to the parent class  """  print("This is the parent Class")  # Define the child class that inherits from the parent class  class child(parent):  def fun2(self):  """  This method prints a message indicating that it belongs to the child class  """  print("This is Child CLass")  # Create an instance of the child class and call its methods  object=child()  object.fun1()  object.fun2() |
| OUTPUT- |

PROGRAM -2

AIM- WAPP FOR MULTIPLE INHERITANCE

CODE-

|  |
| --- |
| print("HARSH D")  class parent1():  '''  This is the parent1 class which has a method fun1  '''  def fun1(self):  # Method to print a message  print("This is Parent 1")  class parent2():  '''  This is the parent2 class which has a method fun2  '''  def fun2(self):  # Method to print a message  print("This is Parent 2")  class child(parent1,parent2):  '''  This is the child class inheriting from parent1 and parent2  '''  def fun3(self):  # Method to print a message  print("This is a Child Class Calling Parent 1 & Parent 2")  obj=child()  obj.fun1()  obj.fun2()  obj.fun3() |
| OUTPUT- |

PROGRAM -3

AIM- WAPP FOR MULTILEVEL INHERITANCE

CODE-

|  |
| --- |
| print("HARSH D")  class Grandparent():  """  This is the Grandparent class  """  def fun1(self):  """  This method prints a message for Grandparent  """  print("This is Grandparent")  class Parent(Grandparent):  """  This is the Parent class  """  def fun2(self):  """  This method prints a message for Parent  """  print("This is Parent")  class child(Parent):  """  This is the child class  """  def fun3(self):  """  This method prints a message for child  """  print("This is child")  # Creating objects and calling methods  obj = child()  obj.fun1()  obj.fun2()  obj.fun3()  object = Parent()  object.fun1()  object.fun2()  object1 = Grandparent()  object1.fun1() |
| OUTPUT- |

PROGRAM -4

AIM- WAPP FOR **Hierarchical Inheritance:** INHERITANCE

CODE-

|  |
| --- |
| print("HARSH D")  class Parent():    def fun1(self):      """      This is the fun1 method of the Parent class      """      print("This is parent class")  class Child1(Parent):    def fun2(self):      """      This is the fun2 method of the Child1 class      """      print("This is child1 class")  class Child2(Parent):    def fun3(self):       """       This is the fun3 method of the Child2 class       """       print("This is child2 class")  class Child3(Parent):    def fun4(self):      """      This is the fun4 method of the Child3 class      """      print("This is child3 class")    object=Child1()  object.fun1()  object.fun2()  object=Child2()  object.fun1()  object.fun3()  object=Child3()  object.fun1()  object.fun4() |
| OUTPUT- |

PROGRAM -5

AIM- WAPP FOR **Hybrid Inheritance:** INHERITANCE

CODE-

|  |
| --- |
| # This code defines a Parent class and multiple Child classes that inherit from the Parent class.  print("HARSH D")  class Parent:    def fun1(self):      """      This method represents the functionality of the parent class.      """      print("This is parent class")  class Child1(Parent):    def fun2(self):      """      This method represents the functionality of the child1 class.      """      print("This is child1 class")  class Child2(Parent):    def fun3(self):      """      This method represents the functionality of the child2 class.      """      print("This is child2 class")  class Child3(Parent):    def fun4(self):      """      This method represents the functionality of the child3 class.      """      print("This is child3 class")  class HybridChild(Child1, Child2):    """    This class inherits from both Child1 and Child2 classes.    """    pass  object = HybridChild()  object.fun1()  object.fun2()  object.fun3() |
| OUTPUT- |

PROGRAM -6

AIM- WAPP FOR SUPER FUNCTION

CODE- OUTPUT-

|  |
| --- |
| class employee():    """    This class represents an employee with attributes name, id, and address    """    print("HARSH D")    def \_\_init\_\_(self,name,id,address):      self.name = name  # Initialize the name      self.id = id      # Initialize the id      self.address = address  # Initialize the address  class tempemployee(employee):    """    This class represents a temporary employee, inherits from employee, and adds the attribute email    """    def \_\_init\_\_(self,name,id,address,email):      super().\_\_init\_\_(name,id,address)  # Call the base class constructor      self.email = email  # Initialize the email  object = tempemployee("Ram","1","Bangalore","oqibz@example.com")  print(object.name)  # Print the name  print(object.id)    # Print the id  print(object.address)  # Print the address  print(object.email)  # Print the email  object.id = "2"    # Update the id  object.name = "Shyam"  # Update the name  object.address = "Chennai"  # Update the address  object.email = "tugrp@example.com"  # Update the email  print("Name = ",object.name)  # Print the updated name  print("ID = ",object.id)  # Print the updated id  print("Address = ",object.address)  # Print the updated address  print("Email = ",object.email)  # Print the updated email |

PROGRAM -7

AIM- WAPP FOR SUPER FUNCTION WITH HAVING A SHAPE CLASS ,CIRCLE CLASS,RECTANGLE CLASS WITH THEIR PROPER CALCULATION [INPUT FROM USER].

CODE-

|  |
| --- |
| """  This program defines classes to represent shapes and calculates their areas.  """  # Displaying a name  print("HARSH D")  # Defining the shape class  class shape():    def \_\_init\_\_(self, color):      """      Initializes the shape with a color      """      self.color = input("Enter the color: ")  # Get user input for color    def calculate\_area(self):      """      Calculates the area of the shape (not implemented in this base class)      """      pass  # Defining the circle class which inherits from shape  class circle(shape):    def \_\_init\_\_(self, color, radius):      """      Initializes the circle with a color and radius      """      super().\_\_init\_\_(color)  # Call the shape class constructor      self.radius = int(input("Enter the radius for circle: "))  # Get user input for radius    def calculate\_area(self):      """      Calculates the area of the circle      """      self.area = 3.14 \* self.radius \* self.radius  # Calculate the area of the circle  # Defining the Rectangle class which inherits from shape  class Rectangle(shape):    def \_\_init\_\_(self, color, length, width):      """      Initializes the rectangle with a color, length, and width      """      super().\_\_init\_\_(color)  # Call the shape class constructor      self.length = int(input("Enter the length for rectangle: "))  # Get user input for length      self.width = int(input("Enter the width for rectangle: "))  # Get user input for width    def calculate\_area(self):      """      Calculates the area of the rectangle      """      self.area = self.length \* self.width  # Calculate the area of the rectangle  # Defining the Triangle class which inherits from shape  class Triangle(shape):    def \_\_init\_\_(self, color, base, height):      """      Initializes the triangle with a color, base, and height      """      super().\_\_init\_\_(color)  # Call the shape class constructor      self.base = int(input("Enter the base for triangle: "))  # Get user input for base      self.height = int(input("Enter the height for triangle: "))  # Get user input for height    def calculate\_area(self):      """      Calculates the area of the triangle      """      self.area = 0.5 \* self.base \* self.height  # Calculate the area of the triangle  # Create an instance of circle and calculate its area  object = circle("red", 10)  object.calculate\_area()  print(object.color)  # Print the color  print(object.area)  # Print the area  print(object.radius)  # Print the radius  # Create an instance of rectangle and calculate its area  object = Rectangle("red", 10, 10)  object.calculate\_area()  print(object.color)  # Print the color  print(object.area)  # Print the area  print(object.length)  # Print the length  print(object.width)  # Print the width  # Create an instance of triangle and calculate its area  object = Triangle("red", 10, 10)  object.calculate\_area()  print(object.color)  # Print the color  print(object.area)  # Print the area  print(object.base)  # Print the base  print(object.height)  # Print the height |

OUTPUT-

