```
"""
Python-I Practice Book - Chapter 9 Solutions
Inheritance, Polymorphism, and NumPy

This file contains solutions to all programming questions from Chapter 9
"""

# ============================================================================
# INHERITANCE PROBLEMS
# ============================================================================

# Question 632: Book and CourseBook Hierarchy
print("="*60)
print("Question 632: Book and CourseBook Hierarchy")
print("="*60)

class Book:
    def __init__(self, name, n, authors, publisher, ISBN, year):
        self.name = name
        self.n = n
        self.authors = authors
        self.publisher = publisher
        self.ISBN = ISBN
        self.year = year

    def display(self):
        print(f"Book Name: {self.name}")
        print(f"Number of Authors: {self.n}")
        print(f"Authors: {', '.join(self.authors)}")
        print(f"Publisher: {self.publisher}")
        print(f"ISBN: {self.ISBN}")
        print(f"Year: {self.year}")

class CourseBook(Book):
    def __init__(self, name, n, authors, publisher, ISBN, year, course):
        super().__init__(name, n, authors, publisher, ISBN, year)
        self.course = course

    def display(self):
        super().display()
        print(f"Course: {self.course}")

# Example usage
book1 = CourseBook("Python Programming", 2, ["John Doe", "Jane Smith"],
                   "Tech Publishers", "978-1234567890", 2024, "Computer Science")
book1.display()
print()

# Question 633: Staff Hierarchy
print("="*60)
print("Question 633: Staff Hierarchy")
print("="*60)

class Staff:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display(self):
        print(f"Name: {self.name}")
        print(f"Salary: {self.salary}")

class Teaching(Staff):
    def __init__(self, name, salary, subject):
        super().__init__(name, salary)
        self.subject = subject

    def display(self):
        super().display()
        print(f"Subject: {self.subject}")

class NonTeaching(Staff):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display(self):
        super().display()
        print(f"Department: {self.department}")

teacher = Teaching("Dr. Smith", 75000, "Mathematics")
teacher.display()
print()
staff_member = NonTeaching("John Doe", 45000, "Administration")
staff_member.display()
print()

# Question 634: Student and PhDguide
print("="*60)
print("Question 634: Student and PhDguide")
print("="*60)

class Student:
    def __init__(self, name, email):
        self.name = name
        self.email = email

    def putdata(self):
        print(f"Student Name: {self.name}, Email: {self.email}")

class PhDguide:
    def __init__(self, name, email):
        self.name = name
        self.email = email
        self.students = []

    def putdata(self):
        print(f"Guide Name: {self.name}, Email: {self.email}")
        print("Students:")
        for student in self.students:
            student.putdata()

    def add(self, student):
        self.students.append(student)
        print(f"Student {student.name} added successfully")

    def remove(self, student_name):
        for student in self.students:
            if student.name == student_name:
                self.students.remove(student)
                print(f"Student {student_name} removed successfully")
                return
        print(f"Student {student_name} not found")
```

```
guide = PhDguide("Dr. Johnson", "johnson@university.edu")
s1 = Student("Alice", "alice@student.edu")
s2 = Student("Bob", "bob@student.edu")
guide.add(s1)
guide.add(s2)
guide.putdata()
guide.remove("Alice")
print("\nAfter removal:")
guide.putdata()
print()

# Question 639: Distance Between Two Points
print("="*60)
print("Question 639: Distance Between Two Points")
print("="*60)

import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return math.sqrt(self.x**2 + self.y**2)

    def translate(self, dx, dy):
        return Point(self.x + dx, self.y + dy)

    def reflect_x(self):
        return Point(self.x, -self.y)

    def distance(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

    def display(self):
        print(f"Point({self.x}, {self.y})")

p1 = Point(3, 4)
print(f"Distance from origin: {p1.distance_from_origin():.2f}")
p2 = p1.translate(2, 3)
print("Translated point:", end=" ")
p2.display()
p3 = p2.reflect_x()
print("Reflected point:", end=" ")
p3.display()
p4 = Point(1, 2)
print(f"Distance between points: {p1.distance(p4):.2f}")
print()

# Question 640: Slope Between Two Points
print("="*60)
print("Question 640: Slope Between Two Points")
print("="*60)

class Point2:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def slope(self, other):
        if self.x == other.x:
            return "Undefined (vertical line)"
        return (other.y - self.y) / (other.x - self.x)

p1 = Point2(2, 3)
p2 = Point2(5, 9)
print(f"Slope between points: {p1.slope(p2):.2f}")
print()

# Question 652: Bus Fare Calculator
print("="*60)
print("Question 652: Bus Fare Calculator")
print("="*60)

class Vehicle:
    def __init__(self, name, mileage, seating_capacity):
        self.name = name
        self.mileage = mileage
        self.seating_capacity = seating_capacity

    def fare(self):
        return self.seating_capacity * 100

class Bus(Vehicle):
    def fare(self):
        base_fare = super().fare()
        maintenance = base_fare * 0.10
        return base_fare + maintenance

bus = Bus("School Bus", 12, 50)
print(f"Bus fare: {bus.fare()}")
car = Vehicle("Sedan", 25, 5)
print(f"Car fare: {car.fare()}")
print()

# Question 653: Abstract Shape Class
print("="*60)
print("Question 653: Abstract Shape with Rectangle and Circle")
print("="*60)

from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def calculate_area(self):
        pass

class Rectangle(Shape):
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def calculate_area(self):
        return self.length * self.breadth

    def __gt__(self, other):
        return self.calculate_area() > other.calculate_area()

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
```

```
    def calculate_area(self):
        return math.pi * self.radius ** 2

rect1 = Rectangle(10, 5)
rect2 = Rectangle(8, 6)
circle = Circle(7)
print(f"Rectangle 1 area: {rect1.calculate_area()}")
print(f"Rectangle 2 area: {rect2.calculate_area()}")
print(f"Circle area: {circle.calculate_area():.2f}")
print(f"Rectangle 1 > Rectangle 2: {rect1 > rect2}")
print("\nMRO of Circle class:")
print(Circle.__mro__)
print()

# ============================================================================
# NUMPY PROBLEMS
# ============================================================================

import numpy as np

# Question 644: Create 5x2 Array
print("="*60)
print("Question 644: Create 5x2 Array from Range")
print("="*60)

arr = np.arange(100, 200, 10).reshape(5, 2)
print("5x2 Array:")
print(arr)
print()

# Question 645: Return Third Column
print("="*60)
print("Question 645: Return Third Column")
print("="*60)

sampleArray = np.array([[11, 22, 33], [44, 55, 66], [77, 88, 99]])
print("Original Array:")
print(sampleArray)
print("\nThird column:")
print(sampleArray[:, 2])
print()

# Question 646: Odd Rows and Even Columns
print("="*60)
print("Question 646: Odd Rows and Even Columns")
print("="*60)

sampleArray = np.array([[3, 6, 9, 12],
                        [15, 18, 21, 24],
                        [27, 30, 33, 36],
                        [39, 42, 45, 48],
                        [51, 54, 57, 60]])
print("Original Array:")
print(sampleArray)
result = sampleArray[::2, 1::2]
print("\nOdd rows and even columns:")
print(result)
print()

# Question 647: Sorting Array
print("="*60)
print("Question 647: Sorting Array")
print("="*60)

sampleArray = np.array([[34, 43, 73], [82, 22, 12], [53, 94, 66]])
print("Original Array:")
print(sampleArray)

sorted_by_row = sampleArray[:, sampleArray[1].argsort()]
print("\nSorted by second row:")
print(sorted_by_row)

sorted_by_col = sampleArray[sampleArray[:, 1].argsort()]
print("\nSorted by second column:")
print(sorted_by_col)
print()

# Question 648: Max from Axis 0, Min from Axis 1
print("="*60)
print("Question 648: Max from Axis 0, Min from Axis 1")
print("="*60)

sampleArray = np.array([[34, 43, 73], [82, 22, 12], [53, 94, 66]])
print("Array:")
print(sampleArray)
print("\nMax from axis 0:", np.max(sampleArray, axis=0))
print("Min from axis 1:", np.min(sampleArray, axis=1))
print()

# Question 649: Fahrenheit to Celsius
print("="*60)
print("Question 649: Fahrenheit to Celsius Conversion")
print("="*60)

fahrenheit = np.array([0, 12, 45.21, 34, 99.91, 32])
print("Values in Fahrenheit degrees:")
print(fahrenheit)

celsius = 5 * fahrenheit / 9 - 5 * 32 / 9
print("\nValues in Centigrade degrees:")
print(celsius)

sorted_celsius = np.sort(celsius)
print("\nSorted Celsius:")
print(sorted_celsius)

position = np.where(sorted_celsius == 0.0)
print("\nPosition of 0.0:")
print(position)
print()

# Question 650: Reshape Array
print("="*60)
print("Question 650: Reshape Array")
print("="*60)

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
print("Original 1D Array:")
print(arr)

arr_2d = arr.reshape(3, 4)
print("\n2D Array (3x4):")
```

```
print(arr_2d)

arr_3d = arr.reshape(2, 3, 2)
print("\n3D Array (2x3x2):")
print(arr_3d)
print()

print("="*60)
print("All solutions completed successfully!")
print("="*60)
```