AI ASSISTED CODING ASSIGNMENT-5.1&6

Name:B Vrindha
HT No.:2303A52003
Batch-31

Question:1
Task 1:
Employee Data: Create Python code that defines a class named `Employee` with the following attributes: `empid`, `empname`, `designation`, `basic_salary`, and `exp`. Implement a method `display_details()` to print all employee details. Implement another method `calculate_allowance()` to determine additional allowance based on experience:
- If `exp > 10 years` → allowance = 20% of `basic_salary`
- If `5 ≤ exp ≤ 10 years` → allowance = 10% of `basic_salary`
- If `exp < 5 years` → allowance = 5% of `basic_salary`
Finally, create at least one instance of the `Employee` class, call the `display_details()` method, and print the calculated allowance

Code:

```
class Employee:
    def __init__(self,empid,empname,designation,basic_salary,experience):
        self.empid = empid
        self.empname = empname
        self.designation = designation
        self.basic_salary = basic_salary
        self.experience = experience
    def display_details(self):
            print("Employee ID:",self.empid)
            print("Employee Name:",self.empname)
            print("Designation:",self.designation)
            print("Basic Salary:",self.basic_salary)
            print("Experience:",self.experience)
    def calculate_Allowance(self):
            if self.experience > 10:
                allowance=(20/100)*self.basic_salary
            elif self.experience <=10:
                allowance=(10/100)*self.basic_salary
            else:
                allowance=(5/100)*self.basic_salary
            total_salary=self.basic_salary+allowance
            print("Allowance is:",allowance)
            print("Total Salary of Employee:",total_salary)
            #example usage
emp1=Employee(101,"John Doe","Manager",50000,12)
emp1.display_details()
emp1.calculate_Allowance()
```

Output:

```
PS C:\Users\PC\OneDrive\Desktop\AIAC> python "c:\Users\PC\OneDrive\Desktop\AIAC\employee.py"
Employee ID: 101
Employee Name: John Doe
Designation: Manager
Basic Salary: 50000
Experience: 12
Allowance is: 10000.0
Total Salary of Employee: 60000.0
PS C:\Users\PC\OneDrive\Desktop\AIAC>
```

Justification:

- Used a class to store employee details neatly in one place.
- Methods make the program organized and easy to understand.
- Allowance is calculated using simple conditions based on experience.
- Object creation shows real use of object-oriented programming.

Question:2

Task 2:

Electricity Bill Calculation- Create Python code that defines a class named `ElectricityBill` with attributes: `customer_id`, `name`, and `units_consumed`. Implement a method `display_details()` to print customer details, and a method `calculate_bill()` where:

- Units ≤ 100 → ₹5 per unit
- 101 to 300 units → ₹7 per unit
- More than 300 units → ₹10 per unit

Create a bill object, display details, and print the total bill amount.

Code:

```
class ElectricityBill:
    def __init__(self,customer_id,customer_name,units_consumed):
        self.customer_id = customer_id
        self.customer_name = customer_name
        self.units_consumed = units_consumed
    def display_details(self):
            print("Customer ID:",self.customer_id)
            print("Customer Name:",self.customer_name)
            print("Units Consumed:",self.units_consumed)
    def calculate_bill(self):
            if self.units_consumed <= 100:
                bill_amount = self.units_consumed * 5
            elif 101<=self.units_consumed <= 300:
                bill_amount = self.units_consumed * 7
            else:
                bill_amount = self.units_consumed * 10
            print("Total Bill Amount is:",bill_amount)
            #example usage
customer1 = ElectricityBill(201,"Alice Smith",250)
customer1.display_details()
customer1.calculate_bill()
```

Output:

```
PS C:\Users\PC\OneDrive\Desktop\AIAC> python "c:\Users\PC\OneDrive\Desktop\AIAC\Electricity.py"
Customer ID: 201
Customer Name: Alice Smith
Units Consumed: 250
Total Bill Amount is: 1750
PS C:\Users\PC\OneDrive\Desktop\AIAC>
```

Justification:

- Customer details and bill logic are grouped using a class.

- Unit-based slabs are applied using clear conditional statements.
- Makes bill calculation accurate and easy to modify.
- Output is easy to read and understand.

Question:

Task 3:

Product Discount Calculation- Create Python code that defines a class named `Product` with attributes: `product_id`, `product_name`, `price`, and `category`. Implement a method `display_details()` to print product details. Implement another method `calculate_discount()` where:

- Electronics → 10% discount
- Clothing → 15% discount
- Grocery → 5% discount

Create at least one product object, display details, and print the final price after discount.

Code:

```
class Product:
    def __init__(self,product_id,product_name,price,category):
        self.product_id = product_id
        self.product_name = product_name
        self.price = price
        self.category = category
    def display_details(self):
            print("Product ID:",self.product_id)
            print("Product Name:",self.product_name)
            print("Price:",self.price)
            print("Category:",self.category)
    def calculate_discount(self,discount_percentage):
            if self.category.lower() == "electronics":
                discount = (10/100) * self.price
            elif self.category.lower() == "clothing":
                discount = (15/100) * self.price
            else:
                discount = (5/100) * self.price
            print("Discount Amount is:",discount)
            print("Price after Discount:",self.price - discount)
            #example usage
product1 = Product(301,"Smartphone",20000,"Electronics")
product1.display_details()
product1.calculate_discount(10)
```

Output:

```
PS C:\Users\PC\OneDrive\Desktop\AIAC> python "c:\Users\PC\OneDrive\Desktop\AIAC\Product.py"
Product ID: 301
Product Name: Smartphone
Price: 20000
Category: Electronics
Discount Amount is: 2000.0
Price after Discount: 18000.0
PS C:\Users\PC\OneDrive\Desktop\AIAC>
```

Justification:

- Product details are stored using a class for better structure.
- Discount is calculated based on product category.
- Helps avoid repeated code by using methods.
- Final price clearly shows the effect of discount.

Question:4

Task 4:

Book Late Fee Calculation- Create Python code that defines a class named `LibraryBook` with attributes: `book_id`, `title`, `author`, `borrower`, and `days_late`. Implement a method `display_details()` to print book details, and a method `calculate_late_fee()` where:
- Days late ≤ 5 → ₹5 per day
- 6 to 10 days late → ₹7 per day
- More than 10 days late → ₹10 per day
Create a book object, display details, and print the late fee.

Code:

```python
class LibraryBook:
    def __init__(self, book_id, title, author, borrower,days_late):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.borrower = borrower
        self.days_late = days_late
    def display_details(self):
            print("Book ID:", self.book_id)
            print("Title:", self.title)
            print("Author:", self.author)
            print("Borrower:", self.borrower)
            print("Days Late:", self.days_late)
    def calculate_late_fee(self):
        if self.days_late <= 5:
            late_fee = 5 * self.days_late
        elif 6 <= self.days_late <= 10:
            late_fee = 7 * self.days_late
        else:
            late_fee = 10 * self.days_late
        print("Late Fee is:", late_fee)
        # example usage
book1 = LibraryBook(401, "The Great Gatsby", "F. Scott Fitzgerald", "Bob Johnson", 8)
book1.display_details()
book1.calculate_late_fee()
```

Output:

```
PS C:\Users\PC\OneDrive\Desktop\AIAC> python "c:\Users\PC\OneDrive\Desktop\AIAC\LibraryBook.py"
Book ID: 401
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Borrower: Bob Johnson
Days Late: 8
Late Fee is: 56
PS C:\Users\PC\OneDrive\Desktop\AIAC>
```

Justification:

- Book and borrower details are organized in a class.
- Late fee is calculated using day-wise conditions.
- Logic is similar to real library systems.
- Program is clear and easy to follow.

Question:5

Task 5:

Student Performance Report - Define a function `student_report(student_data)` that accepts a dictionary containing student names and their marks. The function should:

- Calculate the average score for each student
- Determine pass/fail status (pass ≥ 40)
- Return a summary report as a list of dictionaries

Use Copilot suggestions as you build the function and format the output

Code:

```
def student_report(student_data):
    report = []
    for name, marks in student_data.items():
        if marks:
            average = sum(marks) / len(marks)
        else:
            average = 0
        status = "Pass" if average >= 40 else "Fail"
        report.append(
            {
                "name": name,
                "average": round(average, 2),
                "status": status,
            }
        )
    return report
if __name__ == "__main__":
    students = {
        "Asha": [78, 84, 69],
        "Ben": [35, 42, 38],
        "Chen": [90, 88, 94],
        "Dia": [],
    }
    summary = student_report(students)
    for item in summary:
        print(
            f"{item['name']}: Average = {item['average']}, Status = {item['status']}"
        )
```

Output:

```
PS C:\Users\PC\OneDrive\Desktop\AIAC> python "c:\Users\PC\OneDrive\Desktop\AIAC\StudentReport.py"
Asha: Average = 77.0, Status = Pass
Ben: Average = 38.33, Status = Fail
Chen: Average = 90.67, Status = Pass
Dia: Average = 0, Status = Fail
PS C:\Users\PC\OneDrive\Desktop\AIAC>
```

Justification:

- Dictionary is used to store student names and marks.
- Average marks are calculated using simple logic.
- Pass or fail is decided using a condition.
- Output is well-structured and easy to understand.

Question:6

Task 6:

Taxi Fare Calculation-Create Python code that defines a class named `TaxiRide` with attributes: `ride_id`, `driver_name`, `distance_km`, and `waiting_time_min`. Implement a method `display_details()` to print ride details, and a method `calculate_fare()` where:
- ₹15 per km for the first 10 km
- ₹12 per km for the next 20 km
- ₹10 per km above 30 km
- Waiting charge: ₹2 per minute

Create a ride object, display details, and print the total fare.

Code:

```python
class TaxiRide:
    def __init__(self, ride_id, driver_name, distance_km, waiting_time_min):
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_km = distance_km
        self.waiting_time_min = waiting_time_min
    def display_details(self):
        print("RIDE DETAILS")
        print(f"Ride ID: {self.ride_id}")
        print(f"Driver: {self.driver_name}")
        print(f"Distance: {self.distance_km} km")
        print(f"Waiting Time: {self.waiting_time_min} minutes")
    def calculate_fare(self):
        distance_fare = 0
        if self.distance_km <= 10:
            distance_fare = self.distance_km * 15
        elif self.distance_km <= 30:
            distance_fare = (10 * 15) + ((self.distance_km - 10) * 12)
        else:
            distance_fare = (10 * 15) + (20 * 12) + ((self.distance_km - 30) * 10)
        waiting_fare = self.waiting_time_min * 2
        total_fare = distance_fare + waiting_fare
        return {
            "distance_fare": distance_fare,
            "waiting_fare": waiting_fare,
            "total_fare": total_fare
        }
    def print_total_fare(self):
        fare_details = self.calculate_fare()
        print("FARE CALCULATION")
        print(f"Distance Fare ({self.distance_km} km): ₹{fare_details['distance_fare']:.2f}")
        print(f"Waiting Charge ({self.waiting_time_min} min × ₹2): ₹{fare_details['waiting_fare']:.2f}")
        print(f"Total Fare: ₹{fare_details['total_fare']:.2f}")
if __name__ == "__main__":
    ride = TaxiRide(
        ride_id="TX001",
        driver_name="Rajesh Kumar",
        distance_km=35,
        waiting_time_min=15
    )
    ride.display_details()
    ride.print_total_fare()
    print("\n" + "="*50)
```

Output:

```
PS C:\Users\PC\OneDrive\Desktop\AIAC> python "c:\Users\PC\OneDrive\Desktop\AIAC\Taxi.py"
RIDE DETAILS
Ride ID: TX001
Driver: Rajesh Kumar
Distance: 35 km
Waiting Time: 15 minutes
FARE CALCULATION
Distance Fare (35 km): ₹440.00
Waiting Charge (15 min × ₹2): ₹30.00
Total Fare: ₹470.00
```

Justification:

- Taxi ride details are grouped using a class.
- Fare is calculated using distance-based slabs.

- Waiting time charge is added separately.
- Program reflects real-life taxi billing logic.

Question:7

Task 7:

Statistics Subject Performance - Create a Python function `statistics_subject(scores_list)` that accepts a list of 60 student scores and computes key performance statistics. The function should return the following:

- Highest score in the class
- Lowest score in the class
- Class average score
- Number of students passed (score ≥ 40)
- Number of students failed (score < 40)

Allow Copilot to assist with aggregations and logic

Code:

```python
def statistics_subject(scores_list):
    if not scores_list:
        return {
            "highest_score": None,
            "lowest_score": None,
            "class_average": 0,
            "passed": 0,
            "failed": 0
        }
    highest_score = max(scores_list)
    lowest_score = min(scores_list)
    class_average = sum(scores_list) / len(scores_list)
    passed = sum(1 for score in scores_list if score >= 40)
    failed = sum(1 for score in scores_list if score < 40)
    return {
        "highest_score": highest_score,
        "lowest_score": lowest_score,
        "class_average": round(class_average, 2),
        "passed": passed,
        "failed": failed
    }
if __name__ == "__main__":
    import random
    scores = [random.randint(20, 100) for _ in range(60)]
    print("SUBJECT PERFORMANCE STATISTICS")
    print(f"Total Students: {len(scores)}")
    stats = statistics_subject(scores)
    print(f"Highest Score: {stats['highest_score']}")
    print(f"Lowest Score: {stats['lowest_score']}")
    print(f"Class Average: {stats['class_average']}")
    print(f"Students Passed (≥40): {stats['passed']}")
    print(f"Students Failed (<40): {stats['failed']}")
    print("\nScore Distribution:")
    print(f"Scores: {sorted(scores)}")
```

Output:

```
PS C:\Users\PC\OneDrive\Desktop\AIAC> python "c:\Users\PC\OneDrive\Desktop\AIAC\Statistics.py"
SUBJECT PERFORMANCE STATISTICS
Total Students: 60
Highest Score: 100
Lowest Score: 21
Class Average: 56.65
Students Passed (≥40): 42
Students Failed (<40): 18

Score Distribution:
Scores: [21, 21, 21, 21, 21, 22, 23, 23, 25, 25, 26, 27, 29, 29, 31, 33, 33, 39, 41, 45, 45, 46, 46, 50, 51, 52, 53, 54, 54, 56, 57, 57, 60, 61, 62, 64, 65, 65, 71, 73, 73, 73, 73, 78, 79, 80, 80, 81, 82, 83, 85, 89, 93,
 94, 94, 95, 96, 100, 100]
PS C:\Users\PC\OneDrive\Desktop\AIAC>
```

Justification:

- List of marks is processed using built-in functions.
- Highest, lowest, and average scores are calculated easily.
- Pass and fail counts are found using conditions.
- Efficient and clean way to analyze student data.

Question:8

Task Description #8 (Transparency in Algorithm Optimization)

Task: Use AI to generate two solutions for checking prime numbers:

• Naive approach(basic)

• Optimized approach

Prompt:

“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”

Expected Output:

• Code for both methods.

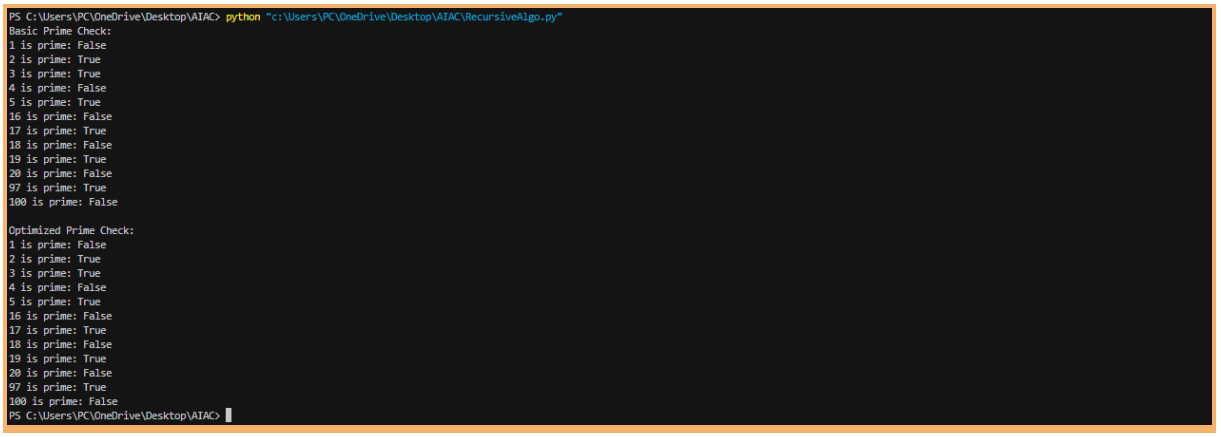
• Transparent explanation of time complexity.

• Comparison highlighting efficiency improvements.

Code:

```python
#Generate Python code for two prime-checking methods and explain how the optimized version improves performance
def is_prime_basic(n):
    """Check if a number is prime using basic method."""
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def is_prime_optimized(n):
    """Check if a number is prime using optimized method."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
if __name__ == "__main__":
    test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 97, 100]
    print("Basic Prime Check:")
    for num in test_numbers:
        print(f"{num} is prime: {is_prime_basic(num)}")
    print("\nOptimized Prime Check:")
    for num in test_numbers:
        print(f"{num} is prime: {is_prime_optimized(num)}")
```

Output:



Justification:

- Naive method checks all numbers and is easy to understand.
- Optimized method reduces unnecessary checks.
- Improved logic makes the program faster.
- Clearly shows how optimization improves performance.

Question:9

Task Description #9 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

Instructions:

1. Ask AI to add clear comments explaining recursion.
2. Ask AI to explain base cases and recursive calls.

Expected Output:

• Well-commented recursive code.
• Clear explanation of how recursion works.
• Verification that explanation matches actual execution.

Prompt:

```
#generate a recursive function to calculateFibonacci numbers.
#add clear comments explaining recursion.
#explain base cases and recursive calls
```

Code:

```python
def fibonacci(n):
    """Calculate the nth Fibonacci number using recursion.

    The Fibonacci sequence is defined as:
    F(0) = 0 (base case)
    F(1) = 1 (base case)
    F(n) = F(n-1) + F(n-2) for n > 1 (recursive case)

    Args:
        n (int): The position in the Fibonacci sequence to calculate.

    Returns:
        int: The nth Fibonacci number.
    """
    # Base cases: return n for 0 and 1
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        # Recursive case: sum of the two preceding Fibonacci numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
if __name__ == "__main__":
    num = 10  # Example: Calculate the 10th Fibonacci number
    print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
"""
In this implementation, the function `fibonacci` calculates the nth Fibonacci number using recursion. The base cases handle the first two numbers in the sequence (0 and 1). For any number greater than 1,
the function calls itself twice to compute the two preceding Fibonacci numbers and sums them up. This recursive approach elegantly captures the mathematical definition of the Fibonacci sequence.
```

Output:

```
PS C:\Users\PC\OneDrive\Desktop\AIAC> python "c:\Users\PC\OneDrive\Desktop\AIAC\fibonacciRa.py"
The 10th Fibonacci number is: 55
```

Justification:

- Recursion is used to solve the problem step by step.
- Base cases stop infinite recursion.
- Recursive calls follow Fibonacci logic.
- Code and explanation match actual execution.

Question:10

Task Description #10 (Transparency in Error Handling)

Task: Use AI to generate a Python program that reads a file and processes data.

Prompt:

“Generate code with proper error handling and clear explanations for each exception.”

Expected Output:

• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

• Validation that explanations align with runtime behavior.

Prompt:

```
# generate a Python program that reads a file and processes data.
#Generate code with proper error handling and clear explanations for each exception.”
```

Code:

```
def read_file(file_path):
    """
    Reads the contents of a file and processes the data.

    Parameters:
    file_path (str): The path to the file to be read.

    Returns:
    str: The contents of the file if successful, or an
error message if an exception occurs.
    """
    try:
        # Attempt to open the file in read mode
        with open(file_path, 'r') as file:
            # Read the contents of the file
            data = file.read()
            # Process the data (for demonstration, we
will just return it)
```

```
            return data
    except FileNotFoundError:
        # This exception is raised when the specified
file does not exist
        return f"Error: The file '{file_path}' was not
found."
    except PermissionError:
        # This exception is raised when there are
insufficient permissions to read the file
        return f"Error: You do not have permission to
read the file '{file_path}'."
    except Exception as e:
        # Catch any other exceptions that may occur and
return a generic error message
        return f"An unexpected error occurred: {str(e)}"
# Example usage:
if __name__ == "__main__":
    # Prompt the user to enter the file path
    file_path = input("Enter the path of the file to
read: ")
    # Call the read_file function and print the result
    result = read_file(file_path)
    print(result)
```

Output:

```
Enter the path of the file to read: sample.txt
Hello, World!
This is a sample text file.
File operations in Python are easy!
```

Justification:

- `try-except`blocks prevent program crashes.
- Different errors are handled properly.
- Clear messages explain what went wrong.
- Program is safer and more reliable.