AI Assistant Coding

Assignment 5.1 & 6

Name : N. Eshwar **HT. No :** 2303A52072 **Batch:** 32

Q1. 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, employee_id, name, designation,
basic_salary,experience):
        self.employee_id = employee_id
        self.name = name
        self.designation = designation
        self.basic_salary = basic_salary
        self.experience = experience
    def calculate_salary(self):
        pass
    def display_details(self):
        print(f"Employee ID: {self.employee_id}")
        print(f"Name: {self.name}")
        print(f"Designation: {self.designation}")
        print(f"Basic Salary: ${self.basic_salary}")
        print(f"Experience: {self.experience} years")
    def calculate_allowance(self):
        if(self.experience>10):
            allowance = 0.20*self.basic_salary
        elif(self.experience>=5 and self.experience<=10):
```

```
            allowance = 0.10*self.basic_salary
        else:
            allowance = 0.05*self.basic_salary
        total_salary = self.basic_salary + allowance
        print(f"Allowance based on experience ({self.experience} years):
${allowance}")
        print(f"Total Salary of the Employee {self.name} is :
${total_salary}")

# Example usage:
emp = Employee(101, "Alice Smith", "Software Engineer", 80000, 7)
emp.display_details()
emp.calculate_allowance()
```

Output:

```
Employee ID: 101
Name: Alice Smith
Designation: Software Engineer
Basic Salary: $80000
Experience: 7 years
Allowance based on experience (7 years): $8000.0
Total Salary of the Employee Alice Smith is : $88000.0
```

Q2. 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 Electricity_Bill:
    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 calculate_bill(self):
```

```
        if self.units_consumed <= 100:
            rate = 5
        elif self.units_consumed>=101 and self.units_consumed <= 300:
            rate = 7
        else:
            rate = 10
        print(self.units_consumed * rate)
    def display_details(self):
        print(f"Customer Name: {self.customer_name}")
        print(f"Customer ID: {self.customer_id}")
        print(f"Units Consumed: {self.units_consumed}")

# Example usage:
bill = Electricity_Bill("C123", "John Doe", 350)
bill.display_details()
bill.calculate_bill()
```

Output:

```
Customer Name: John Doe
Customer ID: C123
Units Consumed: 350
3500
```

Q3. 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_discount:
    def __init__(self, product_id, product_name,price, category):
        self.product_name = product_name
        self.product_id = product_id
        self.price = price
        self.category = category
    def product_info(self):
        print(f"Product ID: {self.product_id}, Name: {self.product_name},
Price: ${self.price}, Category: {self.category}")
    def calculate_discount(self):
        if self.category.lower() == "electronics":
            discount = 0.10
        elif self.category.lower() == "clothing":
            discount = 0.15
        elif self.category.lower() == "groceries":
            discount = 0.05
        else:
            discount = 0.0
        discounted_price = self.price * (1 - discount)
        return discounted_price
# Example usage:
if __name__ == "__main__":
    product = product_discount(101, "Smartphone", 699.99, "Electronics")
    product.product_info()
    new_price = product.calculate_discount()
    print(f"Discounted Price: ${new_price:.2f}")
```

Output:

```
Product ID: 101, Name: Smartphone, Price: $699.99, Category: Electronics
Discounted Price: $629.99
```

Q4. 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:

```
class LibraryBook:
    def __init__(self, book_id, title, author, borroer, days_late):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.borroer = borroer
        self.days_late = days_late
    def book_info(self):
        print(f"Book ID: {self.book_id}, Title: {self.title}, Author:
{self.author}, Borrower: {self.borroer}, Days Late: {self.days_late}")
    def calculate_late_fee(self):
        if self.days_late <= 5:
            fee_per_day = 5
        elif 6<=self.days_late<=10:
            fee_per_day = 7
        else:
            fee_per_day = 10
        total_fee = self.days_late * fee_per_day
        print(f"Total Late Fee: ${total_fee}")
# Example usage:
if __name__ == "__main__":
    book = LibraryBook(202, "The Great Gatsby", "F. Scott Fitzgerald", "John
Doe", 8)
    book.book_info()
    book.calculate_late_fee()
```

Output:

```
Book ID: 202, Title: The Great Gatsby, Author: F. Scott Fitzgerald, Borrower:
John Doe, Days Late: 8
Total Late Fee: $56
```

Q5. 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):
    result = {}
    for name, marks in student_data.items():
        if not marks:
            result[name] = {
                "Average": 0,
                "Status": "Fail"
            }
            continue
        average = sum(marks) / len(marks)
        if average>=40:
            result[name] = {
                "Average": average,
                "Status": "Pass"
            }
        else:
            result[name] = {
                "Average": average,
                "Status": "Fail"
            }

    return result


student_marks = {
    "Alice": [45, 50, 55],
    "Bob": [30, 35],
    "Charlie": [50, 60, 40]
}
result = student_report(student_marks)
print("Student Averages:", result)
```

Output:

```
Student Averages: {'Alice': {'Average': 50.0, 'Status': 'Pass'}, 'Bob':
{'Average': 32.5, 'Status': 'Fail'}, 'Charlie': {'Average': 50.0, 'Status':
'Pass'}}
```

Q6. 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:

```
class TaxiRide:
    def __init__(self,ride_id, driver_name, distance_km, wait_time_min):
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_km = distance_km
        self.wait_time_min = wait_time_min
    def display_info(self):
        print(f"Ride ID: {self.ride_id}")
        print(f"Driver Name: {self.driver_name}")
        print(f"Distance (km): {self.distance_km}")
        print(f"Wait Time (min): {self.wait_time_min}")
    def calculate_fare(self):
        fare = 0
        if(self.distance_km <=10):
            fare = fare + (self.distance_km - 10) * 15
        elif(self.distance_km<=30):
            fare+= 10 * 15
            fare += (self.distance_km - 10) * 12
        else:
            fare += 10 * 15
            fare += 20 * 12
            fare += (self.distance_km - 30) * 10
        fare += self.wait_time_min * 2
        print(f"Total Fare: {fare} units")
# Example usage:
ride = TaxiRide(101, "John Doe", 25, 5)
ride.display_info()
ride.calculate_fare()
```

Output:

```
Ride ID: 101
Driver Name: John Doe
Distance (km): 25
Wait Time (min): 5
Total Fare: 340 units
```

Q7. 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:

```
def statistics_subject(scores_list):
    results = {}
    for subject, scores in scores_list.items():
        if not scores:
            results[subject] = {
                'average': None,
                'highest': None,
                'lowest': None
            }
            continue

        average_score = sum(scores) / len(scores)
        highest_score = max(scores)
        lowest_score = min(scores)
        pass_students = 0
        failed_students = 0
        for score in scores:
            if score >= 40:
                pass_students += 1
            else:
                failed_students += 1

        results[subject] = {
            'average': average_score,
            'highest': highest_score,
```

```
            'lowest': lowest_score,
            'Number of Pass Students': pass_students,
            'Number of Failed Students': failed_students
        }
    return results
# Example usage
scores_data = {
    'Math': [85, 78, 92, 45, 33, 67, 89, 90, 100, 56, 73, 49, 38, 77, 84, 91,
60, 55, 44, 39, 72, 81, 69, 88, 95, 40, 66, 74, 82, 87, 93, 50, 61, 79, 83,
94, 71, 64, 57, 46, 75, 68, 54, 42, 41, 62, 63, 59, 58, 52, 51, 53, 47, 43,
36, 37, 34, 35, 30, 29],
    'Science': [88, 90, 76, 54, 39, 67, 85, 92, 100, 45, 73, 81, 60, 49, 38,
77, 84, 91, 70, 55, 44, 33, 72, 79, 68, 87, 95, 40, 66, 74, 82, 89, 50, 61,
78, 83, 94, 71, 64, 57, 46, 75, 69, 54, 42, 41, 62, 63, 59, 58, 52, 51, 53,
47, 43, 36, 37, 34, 35, 30, 29],
    'English': [70, 80, 65, 50, 40, 30, 90, 85, 75, 95, 60, 55, 45, 35, 25,
100, 78, 82, 88, 92, 68, 58, 48, 38, 28, 22, 66, 72, 74, 84, 86, 94, 52, 54,
56, 64, 62, 44, 42, 34, 32, 26, 24, 20, 18, 16, 14, 12, 10, 8, 6, 4, 2]
}

performance_stats = statistics_subject(scores_data)
for subject, stats in performance_stats.items():
    print(f"Subject: {subject}")
    for stat_name, value in stats.items():
        print(f"  {stat_name}: {value}")
    print()
```

Output:

```
Subject: Math
  average: 62.766666666666666
  highest: 100
  lowest: 29
  Number of Pass Students: 51
  Number of Failed Students: 9

Subject: Science
  average: 62.57377049180328
  highest: 100
  lowest: 29
  Number of Pass Students: 52
  Number of Failed Students: 9

Subject: English
  average: 50.528301886792455
  highest: 100
  lowest: 2
  Number of Pass Students: 33
```

```
  Number of Failed Students: 20
```

Q8. 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:

```
'''
Generate a well commentd  python code for checking if a number is prime or
not. i naive approach(basic) and optimized approach.
'''


import time


def is_prime_naive(n):
    """
    Check if a number is prime using a naive approach.

    A prime number is a natural number greater than 1 that cannot be formed by
multiplying
    two smaller natural numbers. The naive approach checks for factors from 2
to n-1.

    Parameters:
    n (int): The number to check for primality.

    Returns:
    bool: True if n is prime, False otherwise.
    """
```

```
    if n <= 1:
        return False  # Numbers less than or equal to 1 are not prime
    for i in range(2, n):
        if n % i == 0:
            return False  # Found a factor, so n is not prime
    return True  # No factors found, so n is prime
def is_prime_optimized(n):
    """
    Check if a number is prime using an optimized approach.
    This approach reduces the number of checks needed by only testing for
factors
    up to the square root of n and skipping even numbers after checking for 2.

    Parameters:
    n (int): The number to check for primality.

    Returns:
    bool: True if n is prime, False otherwise.
    """
    if n <= 1:
        return False  # Numbers less than or equal to 1 are not prime
    if n <= 3:
        return True  # 2 and 3 are prime numbers
    if n % 2 == 0 or n % 3 == 0:
        return False  # Multiples of 2 and 3 are not prime
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False  # Found a factor, so n is not prime
        i += 6
    return True  # No factors found, so n is prime
# Example usage:
if __name__ == "__main__":
    number = 29
    start_time = time.time()
    print(f"Naive approach: Is {number} prime? {is_prime_naive(number)}")
    print(f"Naive approach took {time.time() - start_time} seconds")
    start_time = time.time()
    print(f"Optimized approach: Is {number} prime?
{is_prime_optimized(number)}")
    print(f"Optimized approach took {time.time() - start_time} seconds")
```

Output:

```
Naive approach: Is 29 prime? True
Naive approach took 0.0002608299255371094 seconds
Optimized approach: Is 29 prime? True
Optimized approach took 6.699562072753906e-05 seconds
```

Q9. 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

Code:

```
'''
genarte a well commentd python code to calculate febonacci series  using
recursion
'''
def fibonacci(n):
    """
    Calculate the nth Fibonacci number using recursion.

    Parameters:
    n (int): The position in the Fibonacci series to calculate.

    Returns:
    int: The nth Fibonacci number.
    """
    # Base case: the first two Fibonacci numbers are 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)
# Example usage:
num_terms = 10  # Number of terms in the Fibonacci series to generate
fibonacci_series = [fibonacci(i) for i in range(num_terms)]
print("Fibonacci Series up to", num_terms, "terms:", fibonacci_series)
```

Output:

```
Fibonacci Series up to 10 terms: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Q10. 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.

Code:

```
'''
Generate a well commented python code that reads a text file handiles errors
and print clear explanations for each error.
'''
def read_file(file_path):
    """
    Reads the content of a text file and handles potential errors.

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

    Returns:
    None
    """
    try:
        # Attempt to open the file in read mode
        with open(file_path, 'r') as file:
            content = file.read()
```

```
            print("File content successfully read:")
            print(content)

    except FileNotFoundError:
        # Handle the case where the file does not exist
        print(f"Error: The file at '{file_path}' was not found. Please check
the file path and try again.")

    except IsADirectoryError:
        # Handle the case where a directory is provided instead of a file
        print(f"Error: The path '{file_path}' is a directory, not a file.
Please provide a valid file path.")

    except PermissionError:
        # Handle the case where there are insufficient permissions to read the
file
        print(f"Error: You do not have permission to read the file at
'{file_path}'. Please check your permissions.")

    except Exception as e:
        # Handle any other unexpected exceptions
        print(f"An unexpected error occurred: {e}")
# Example usage
if __name__ == "__main__":
    file_path = "example.txt"  # Replace with your file path
    read_file(file_path)
```

Output:

```
File content successfully read:
This is an example text file.
It contains multiple lines of text.
Enjoy reading!
```