Assigiment 5.1

Hallticket no:2303A510A6

Batch no:06

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:

ass9ai.py > ...

```python
class Employee:
    def __init__(self, empid, empname, designation, basic_salary, exp):
        self.empid = empid
        self.empname = empname
        self.designation = designation
        self.basic_salary = basic_salary
        self.exp = exp

    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 (years):", self.exp)

    def calculate_allowance(self):
        if self.exp > 10:
            allowance = 0.20 * self.basic_salary
        elif 5 <= self.exp <= 10:
            allowance = 0.10 * self.basic_salary
        else:
            allowance = 0.05 * self.basic_salary
        return allowance


# Creating an instance of Employee
emp1 = Employee(101, "Alice Johnson", "Software Engineer", 60000, 8)

# Display employee details
emp1.display_details()

# Calculate and print allowance
allowance = emp1.calculate_allowance()
print("Allowance:", allowance)

```

Explanation:

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:

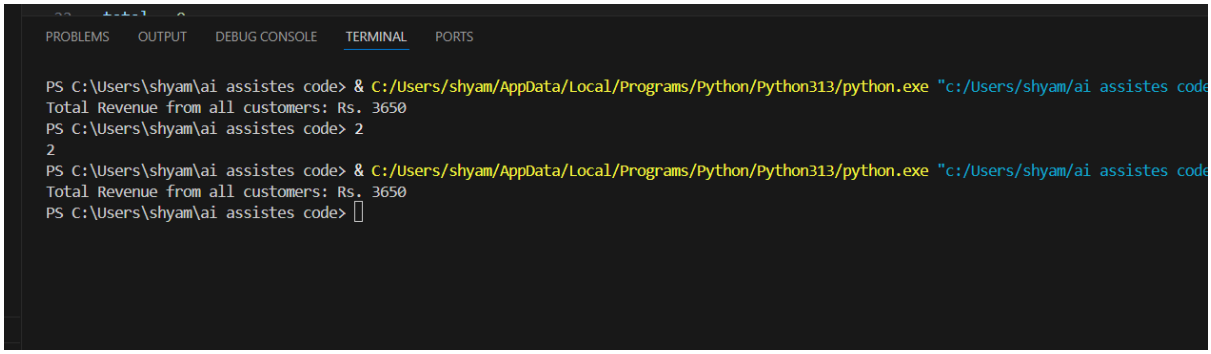
- Units ≤ 100 → ₹5 per unit
- 101 to 300 units → ₹7 per unit
- More than 300 units → ₹10 per unit

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

Code:

```python
class Bill_electricity:
    def __init__(self, customer_id, name, units_consumed):
        self.customer_id = customer_id
        self.name = name
        self.units_consumed = units_consumed

    def calculate_bill(self):
        if self.units_consumed <= 100:
            return self.units_consumed * 5
        elif self.units_consumed <= 300:
            return (100 * 5) + (self.units_consumed - 100) * 7
        else:
            return (100 * 5) + (200 * 7) + (self.units_consumed - 300) * 10


bills = [
    Bill_electricity(1, "Alice", 150),
    Bill_electricity(2, "Bob", 80),
    Bill_electricity(3, "Charlie", 350)
]

total = 0
for bill in bills:
    total += bill.calculate_bill()

print("Total Revenue from all customers: Rs.", total)

```

Output:



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

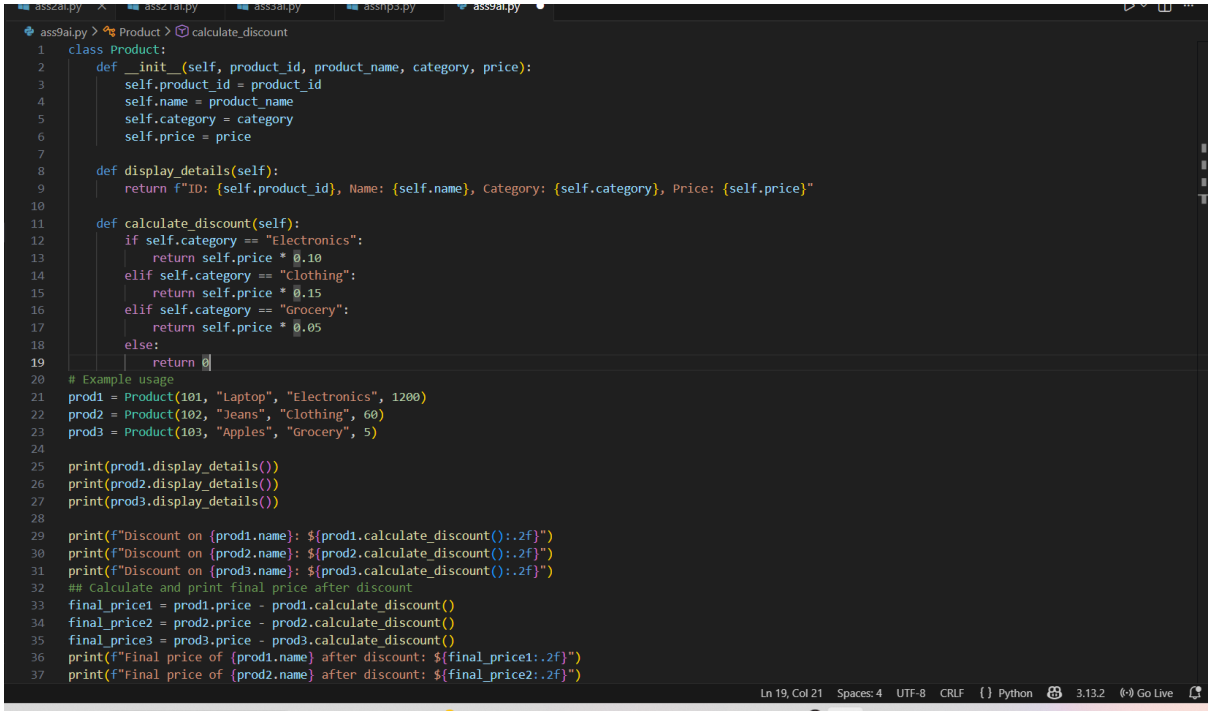
- Clothing → 15% discount

- Grocery → 5% discount

Create at least one product object, display details, and print the final

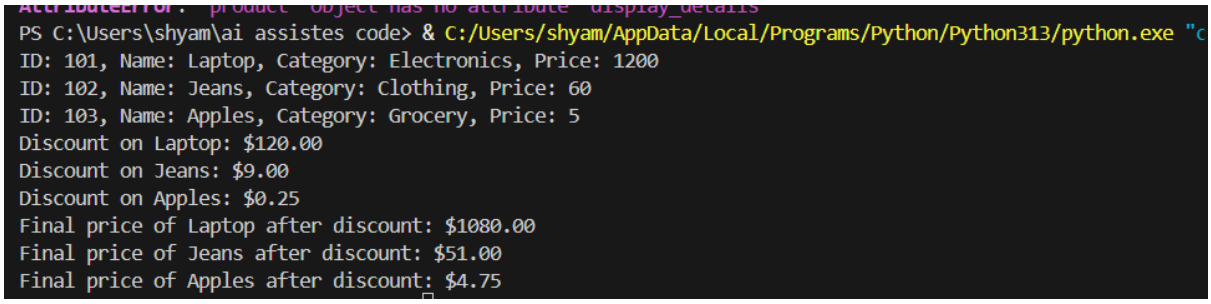
price after discount.

Code:



```python
class Product:
    def __init__(self, product_id, product_name, category, price):
        self.product_id = product_id
        self.name = product_name
        self.category = category
        self.price = price

    def display_details(self):
        return f"ID: {self.product_id}, Name: {self.name}, Category: {self.category}, Price: {self.price}"

    def calculate_discount(self):
        if self.category == "Electronics":
            return self.price * 0.10
        elif self.category == "Clothing":
            return self.price * 0.15
        elif self.category == "Grocery":
            return self.price * 0.05
        else:
            return 0
# Example usage
prod1 = Product(101, "Laptop", "Electronics", 1200)
prod2 = Product(102, "Jeans", "Clothing", 60)
prod3 = Product(103, "Apples", "Grocery", 5)

print(prod1.display_details())
print(prod2.display_details())
print(prod3.display_details())

print(f"Discount on {prod1.name}: ${prod1.calculate_discount():.2f}")
print(f"Discount on {prod2.name}: ${prod2.calculate_discount():.2f}")
print(f"Discount on {prod3.name}: ${prod3.calculate_discount():.2f}")
## Calculate and print final price after discount
final_price1 = prod1.price - prod1.calculate_discount()
final_price2 = prod2.price - prod2.calculate_discount()
final_price3 = prod3.price - prod3.calculate_discount()
print(f"Final price of {prod1.name} after discount: ${final_price1:.2f}")
print(f"Final price of {prod2.name} after discount: ${final_price2:.2f}")
```

Output:



```
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c
ID: 101, Name: Laptop, Category: Electronics, Price: 1200
ID: 102, Name: Jeans, Category: Clothing, Price: 60
ID: 103, Name: Apples, Category: Grocery, Price: 5
Discount on Laptop: $120.00
Discount on Jeans: $9.00
Discount on Apples: $0.25
Final price of Laptop after discount: $1080.00
Final price of Jeans after discount: $51.00
Final price of Apples after discount: $4.75
```

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 libarybook:
    def __init__(self, book_id, author,borrower, day_late):
        self.book_id = book_id
        self.author = author
        self.borrower = borrower
        self.day_late = day_late

    def display_details(self):
        return f"Book ID: {self.book_id}, Author: {self.author}, Borrower: {self.borrower}, Days Late: {self.day_late}"
    def calculate_fine(self):
        if self.day_late <= 5:
            return 5.00
        elif 6 <= self.day_late <= 10:
            return 7.00
        elif self.day_late > 10:
            return 10.00

        fine_per_day = 0.50
        total_fine = self.day_late * fine_per_day
        return total_fine
# Example usage:
book1 = libarybook("B001", "George Orwell", "Alice", 4)
print(book1.display_details())
print(f"Fine for {book1.borrower}: ${book1.calculate_fine():.2f}")

# Display book details
book1.display_details()
#Calculate and print late fee
print(f"Late Fee: ${book1.calculate_fine():.2f}")
```

Output:

```
        ^^^^^^^^^^^^^^^^^^
AttributeError: 'libarybook' object has no attribute 'get_book_info'
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass9ai.p
Book ID: B001, Author: George Orwell, Borrower: Alice, Days Late: 4
Fine for Alice: $5.00
Late Fee: $5.00
PS C:\Users\shyam\ai assistes code>
```

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

Code:

```python
class Student_report:
    def __init__(self, name,marks ,Grade):
        self.name = name
        self.marks = marks
        self.Grade = Grade


    def display_report(self):
        print(f"Student Name: {self.name}")
        print(f"Age: {self.marks}")
        print(f"Grade: {self.Grade}")

        def caculate_average(self):
            return sum(self.marks) / len(self.marks)
        if self.marks >= 40:
            print("Grade : pass")
        else:
            print("Grade : fail")
# Example usage
student1 = Student_report("Alice", 85, 'A')
student1.display_report()
student2 = Student_report("Bob", 35, 'F')
student2.display_report()
```

Output:

```
SyntaxError: invalid syntax
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass9ai
Student Name: Alice
Age: 85
Grade: A
Grade : pass
Student Name: Bob
Age: 35
Grade: F
Grade : fail
PS C:\Users\shyam\ai assistes code>
```

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 ID:", self.ride_id)
        print("Driver Name:", self.driver_name)
        print("Distance (km):", self.distance_km)
        print("Waiting Time (min):", self.waiting_time_min)

    def calculate_fare(self):
        distance = self.distance_km
        fare = 0

        if distance <= 10:
            fare += distance * 15
        elif distance <= 30:
            fare += (10 * 15) + (distance - 10) * 12
        else:
            fare += (10 * 15) + (20 * 12) + (distance - 30) * 10

        waiting_charge = self.waiting_time_min * 2
        total_fare = fare + waiting_charge

        return total_fare

# Creating a TaxiRide object
ride1 = TaxiRide(501, "Suresh", 35, 8)

# Display ride details
ride1.display_details(self)

# Calculate and print total fare
```

Output:

```
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass9ai.py"
Ride ID: 501
Driver Name: Suresh
Distance (km): 35
Waiting Time (min): 8
Total Fare: ₹ 456
```

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

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.

Task

Code:

ass9ai.py > ...

```python
def statistics_subject(scores_list):
    highest = max(scores_list)
    lowest = min(scores_list)
    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,
        "Lowest Score": lowest,
        "Average Score": round(average, 2),
        "Students Passed": passed,
        "Students Failed": failed
    }


# Example usage (sample list of 60 scores)
scores = [
    55, 72, 38, 49, 60, 45, 80, 33, 41, 67,
    52, 59, 44, 29, 75, 63, 48, 50, 36, 90,
]

report = statistics_subject(scores)

# Display results
for key, value in report.items():
    print(f"{key}: {value}")

```

Output:

```
.py
Highest Score: 90
Lowest Score: 29
Average Score: 54.3
Students Passed: 16
Students Failed: 4
PS C:\Users\shyam\ai assistes code>
```

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.”

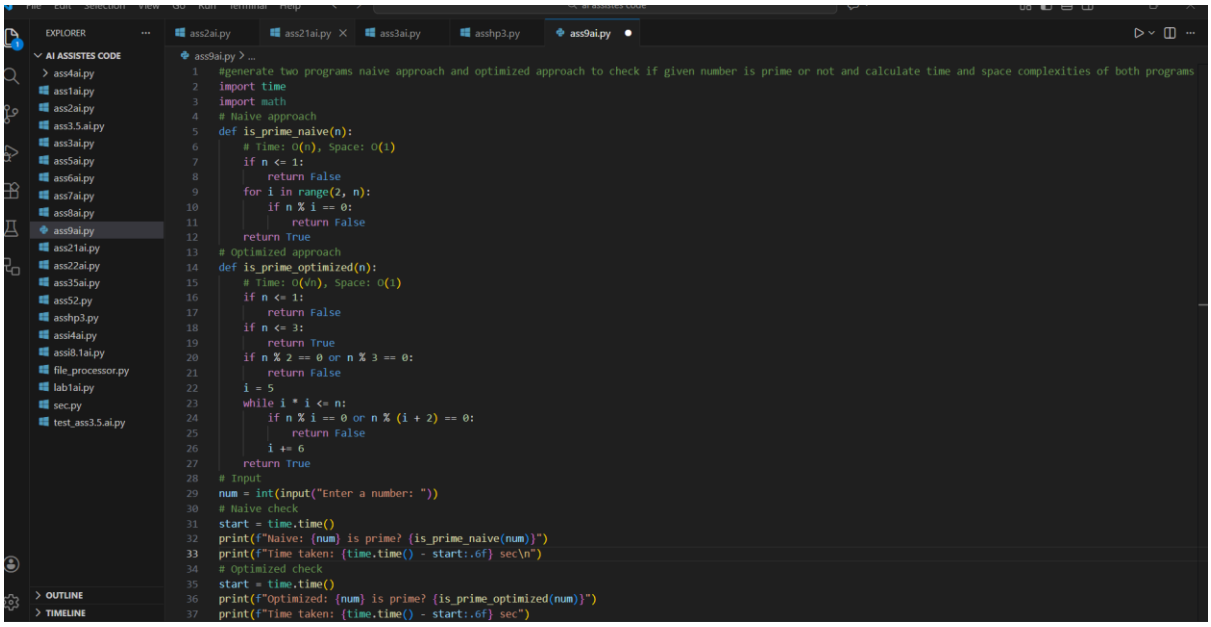
Expected Output:

• Code for both methods.

• Transparent explanation of time complexity.

• Comparison highlighting efficiency improvements.

Code:



Output:

```
Optimized Approach: Is 29 prime? True
NameError: name 'time' is not defined. Did you forget to import 'time'?
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes
Naive Approach: Is 29 prime? True
Time taken (Naive): 4.5299530029296875e-06 seconds
Optimized Approach: Is 29 prime? True
Naive Approach: Is 29 prime? True
Time taken (Naive): 4.5299530029296875e-06 seconds
Optimized Approach: Is 29 prime? True
Optimized Approach: Is 29 prime? True
Time taken (Optimized): 2.384185791015625e-06 seconds
PS C:\Users\shyam\ai assistes code>
```

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:

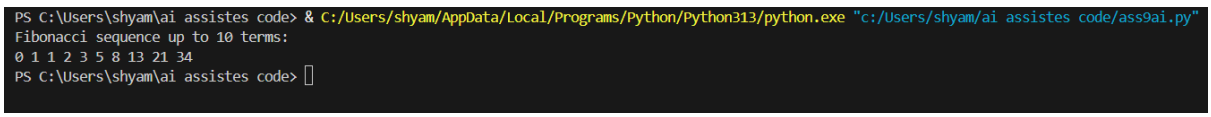
• Well-commented recursive code.

• Clear explanation of how recursion works.

• Verification that explanation matches actual execution

Code:

Output:



Explanation:

The program reads a file safely using a function and processes its contents line by line. Each line is cleaned of spaces and empty lines are skipped. Valid lines are converted into numbers, and invalid lines are ignored with a warning. The program handles common errors: if the file doesn't exist, if there are permission issues, or if unexpected errors occur, it shows clear messages instead of crashing. Finally, the processed numbers are displayed as a list.

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:

```python
# Function to read a file and process its data
#Generate code with proper error handling and clear explanations for each exception.
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            data = file.readlines()
            print("File content successfully read.")
            processed_data = []


            for line in data:
                line = line.strip()
                if line:
                    try:
                        processed_data.append(int(line))
                    except ValueError:
                        print(f"Warning: Skipping invalid data -> '{line}'")

            return processed_data

    except FileNotFoundError:
        print(f"Error: The file at {file_path} was not found.")
    except PermissionError:
        print(f"Error: You do not have permission to read the file at {file_path}.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

file_path = 'example.txt'
file_content = read_file(file_path)
if file_content:
    print("\nProcessed File Content:")
    print(file_content)
else:
    print("\nNo valid data to display.")

```

Output:

```
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass9ai.py"
Error: The file at example.txt was not found.

No valid data to display.
PS C:\Users\shyam\ai assistes code>
```

Explaintion:

This program reads a file and processes its contents safely using a function with proper error handling. It removes empty lines, converts valid lines to numbers, and skips invalid data with a warning. Exceptions like missing files, permission issues, or unexpected errors are handled gracefully, ensuring the program doesn’t crash while providing clear messages to the user. The processed data is returned as a list and displayed.