Lab Aasignment-5.1

Name:D.BHARATH KUMAR

Hallticket:2303A51097

Batch-02

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:
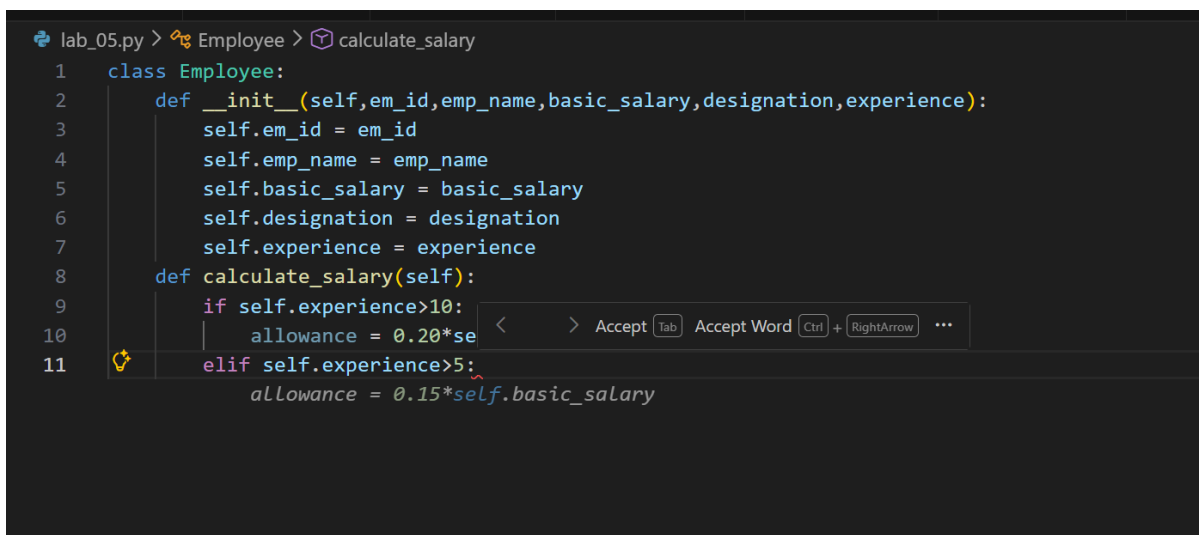
- If `exp > 10 years` → allowance = 20% of `basic_salary`

- 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.

Screenshots:

```
lab_05.py > Employee > calculate_salary
1    class Employee:
2        def __init__(self,em_id,emp_name,basic_salary,designation,experience):
3            self.em_id = em_id
4            self.emp_name = emp_name
5            self.basic_salary = basic_salary
6            self.designation = designation
7            self.experience = experience
8        def calculate_salary(self):
9            if self.experience>10:
10               allowance = 0.20*se     <    >  Accept Tab  Accept Word Ctrl + RightArrow  ...
11           elif self.experience>5:
                 allowance = 0.15*self.basic_salary
```

```python
class Employee:
    def __init__(self,em_id,emp_name,basic_salary,designation,experience):
        self.em_id = em_id
        self.emp_name = emp_name
        self.basic_salary = basic_salary
        self.designation = designation
        self.experience = experience
    def calculate_salary(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
        elif self.experience<5:
            allowance = 0.05*self.basic_salary
        total_salary = self.basic_salary + allowance
        return total_salary
    def display_employee_details(self):
        print(f"Employee ID: {self.em_id}")
        print(f"Employee Name: {self.emp_name}")
        print(f"Designation: {self.designation}")
        print(f"Experience: {self.experience} years")
        print(f"Total Salary: {self.calculate_salary()}")
```

```python
class Employee:
    def __init__(self,em_id,emp_name,basic_salary,designation,experience):
        self.em_id = em_id
        self.emp_name = emp_name
        self.basic_salary = basic_salary
        self.designation = designation
        self.experience = experience
    def calculate_salary(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
        elif self.experience<5:
            allowance = 0.05*self.basic_salary
        total_salary = self.basic_salary + allowance
        return total_salary
    def display_employee_details(self):
        total_salary = self.calculate_salary()
        print(f"Employee ID: {self.em_id}")
        print(f"Employee Name: {self.emp_name}")
        print(f"Designation: {self.designation}")
        print(f"Experience: {self.experience} years")
        print(f"Total Salary: {total_salary}")
# Example usage:
emp1 = Employee(101, "Alice Smith", 50000, "Software Engineer", 6)
emp1.display_employee_details()
emp2 = Employee(102, "Bob Johnson", 70000, "Senior Developer", 12)
emp2.display_employee_details()

```

Code:

```python
class Employee:
    def __init__(self,em_id,emp_name,basic_salary,designation,experience):
```

```python
        self.em_id = em_id
        self.emp_name = emp_name
        self.basic_salary = basic_salary
        self.designation = designation
        self.experience = experience
    def calculate_salary(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
        elif self.experience<5:
            allowance = 0.05*self.basic_salary
        total_salary = self.basic_salary + allowance
        return total_salary
    def display_employee_details(self):
        total_salary = self.calculate_salary()
        print(f"Employee ID: {self.em_id}")
        print(f"Employee Name: {self.emp_name}")
        print(f"Designation: {self.designation}")
        print(f"Experience: {self.experience} years")
        print(f"Total Salary: {total_salary}")
# Example usage:
emp1 = Employee(101, "Alice Smith", 50000, "Software Engineer", 6)
emp1.display_employee_details()
emp2 = Employee(102, "Bob Johnson", 70000, "Senior Developer", 12)
emp2.display_employee_details()
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/One
Drive/Pictures/Desktop/AIAC/lab_05 (1).py"
Employee ID: 101
Employee Name: Alice Smith
Designation: Software Engineer
Experience: 6 years
Total Salary: 55000.0
Employee ID: 102
Employee Name: Bob Johnson
Designation: Senior Developer
Experience: 12 years
Total Salary: 84000.0
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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.

Screenshots:

```python
30    class ElectricityBill:
31        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 = 1.5
              elif self.units_consumed <= 300:
                  rate = 2.5
              else:
                  rate = 4.0
              total_bill = self.units_consumed * rate
              return total_bill
          def display_bill_details(self):
              total_bill = self.calculate_bill()
              print(f"Customer ID: {self.customer_id}")
              print(f"Customer Name: {self.customer_name}")
              print(f"Units Consumed: {self.units_consumed}")
              print(f"Total Bill Amount: {total_bill}")
```

```python
29    """
30    class ElectricityBill:
31        def __init__(self,customer_id,name,units_consumed):
32            self.customer_id = customer_id
33            self.name = name
34            self.units_consumed = units_consumed
35        def calculate_bill(self):
              if self.units_consumed <= 100:
                  bill_amount = self.units_consumed * 5
              elif self.units_consumed <= 300:
                  bill_amount = (100 * 5) + (self.units_consumed - 100) * 7
              else:
                  bill_amount = (100 * 5) + (200 * 7) + (self.units_consumed - 300) * 10
              return bill_amount
```

```
27    emp2 = Employee(102, "Bob Johnson", 70000, "Senior Developer", 12)
28    emp2.display_employee_details()
29    """
30    class ElectricityBill:
31        def __init__(self,customer_id,name,units_consumed):
32            self.customer_id = customer_id
33            self.name = name
34            self.units_consumed = units_consumed
35        def calculate_bill(self):
36            if self.units_consumed <= 100:
37                bill_amount = self.units_consumed * 5
38            elif self.units_consumed <= 300:
39                bill_amount = (100 * 5) + (self.units_consumed - 100) * 7
40            else:
41                bill_amount = (100 * 5) + (200 * 7) + (self.units_consumed - 300) * 10
42            return bill_amount
43        def display_bill_details(self):
44            bill_amount = self.calculate_bill()
45            print(f"Customer ID: {self.customer_id}")
46            print(f"Customer Name: {self.name}")
47            print(f"Units Consumed: {self.units_consumed}")
48            print(f"Total Bill Amount: {bill_amount}")
49    # Example usage:
50    bill1 = ElectricityBill(201, "Charlie Brown", 250)
51    bill1.display_bill_details()
52    bill2 = ElectricityBill(202, "Diana Prince", 350)
53    bill2.display_bill_details()
```

Code:

```python
class ElectricityBill:
    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:
            bill_amount = self.units_consumed * 5
        elif self.units_consumed <= 300:
            bill_amount = (100 * 5) + (self.units_consumed - 100) * 7
        else:
            bill_amount = (100 * 5) + (200 * 7) + (self.units_consumed - 300)
* 10
        return bill_amount
    def display_bill_details(self):
        bill_amount = self.calculate_bill()
        print(f"Customer ID: {self.customer_id}")
        print(f"Customer Name: {self.name}")
        print(f"Units Consumed: {self.units_consumed}")
        print(f"Total Bill Amount: {bill_amount}")
# Example usage:
bill1 = ElectricityBill(201, "Charlie Brown", 250)
```

```
bill1.display_bill_details()
bill2 = ElectricityBill(202, "Diana Prince", 350)
bill2.display_bill_details()
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/One
Drive/Pictures/Desktop/AIAC/lab_05 (1).py"
Customer ID: 201
Customer Name: Charlie Brown
Units Consumed: 250
Total Bill Amount: 1550
Customer ID: 202
Customer Name: Diana Prince
Units Consumed: 350
Total Bill Amount: 2400
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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.

Screenshots:

```python
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 apply_discount(self,discount_percentage):
        if self.category.lower() == "electronics":
            discount = 0.10 * self.price
        elif self.category.lower() == "clothing":
            discount = 0.15 * self.price
        elif self.category.lower() == "groceries":
            discount = 0.05 * self.price
        else:
            discount = 0
        discounted_price = self.price - discount
        return discounted_price
    def display_product_details(self,discount_percentage):
```

```python
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 apply_discount(self,discount_percentage):
        if self.category.lower() == "electronics":
            discount = 0.10 * self.price
        elif self.category.lower() == "clothing":
            discount = 0.15 * self.price
        elif self.category.lower() == "groceries":
            discount = 0.05 * self.price
        else:
            discount = 0
        discounted_price = self.price - discount
        return discounted_price
    def display_product_details(self,discount_percentage):
        discounted_price = self.apply_discount(discount_percentage)
        print(f"Product ID: {self.product_id}")
        print(f"Product Name: {self.product_name}")
        print(f"Category: {self.category}")
        print(f"Original Price: {self.price}")
        print(f"Discounted Price: {discounted_price}")
```

```python
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 apply_discount(self,discount_percentage):
        if self.category.lower() == "electronics":
            discount = 0.10 * self.price
        elif self.category.lower() == "clothing":
            discount = 0.15 * self.price
        elif self.category.lower() == "groceries":
            discount = 0.05 * self.price
        else:
            discount = 0
        discounted_price = self.price - discount
        return discounted_price
    def display_product_details(self,discount_percentage):
        discounted_price = self.apply_discount(discount_percentage)
        print(f"Product ID: {self.product_id}")
        print(f"Product Name: {self.product_name}")
        print(f"Category: {self.category}")
        print(f"Original Price: {self.price}")
        print(f"Discounted Price: {discounted_price}")
# Example usage:
prod1 = Product(301, "Smartphone", 15000, "Electronics")
prod1.display_product_details(10)
prod2 = Product(302, "Jeans", 2000, "Clothing")
prod2.display_product_details(15)
prod3 = Product(303, "Rice", 500, "Groceries")
prod3.display_product_details(5)
```

Code:

```python
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 apply_discount(self,discount_percentage):
        if self.category.lower() == "electronics":
            discount = 0.10 * self.price
        elif self.category.lower() == "clothing":
            discount = 0.15 * self.price
        elif self.category.lower() == "groceries":
            discount = 0.05 * self.price
        else:
            discount = 0
        discounted_price = self.price - discount
        return discounted_price
```

```python
    def display_product_details(self,discount_percentage):
        discounted_price = self.apply_discount(discount_percentage)
        print(f"Product ID: {self.product_id}")
        print(f"Product Name: {self.product_name}")
        print(f"Category: {self.category}")
        print(f"Original Price: {self.price}")
        print(f"Discounted Price: {discounted_price}")
# Example usage:
prod1 = Product(301, "Smartphone", 15000, "Electronics")
prod1.display_product_details(10)
prod2 = Product(302, "Jeans", 2000, "Clothing")
prod2.display_product_details(15)
prod3 = Product(303, "Rice", 500, "Groceries")
prod3.display_product_details(5)
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/One
Drive/Pictures/Desktop/AIAC/lab_05 (1).py"
Product ID: 301
Product Name: Smartphone
Category: Electronics
Original Price: 15000
Discounted Price: 13500.0
Product ID: 302
Product Name: Jeans
Category: Clothing
Original Price: 2000
Discounted Price: 1700.0
Product ID: 303
Product Name: Rice
Category: Groceries
Original Price: 500
Discounted Price: 475.0
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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.

Screenshots:

```python
class LibraryBook:
    def __init__(self,book_id,title,author,genre,availability):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.genre = genre
        self.availability = availability
    def check_availability(self):
        return self.availability
    def display_book_details(self):
        availability_status = "Available" if self.availability else "Not Available"
        print(f"Book ID: {self.book_id}")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Genre: {self.genre}")
        print(f"Availability: {availability_status}")
```

```python
class LibraryBook:
    def __init__(self,book_id,title,author,barrower,days_late):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.barrower = barrower
        self.days_late = days_late
    def calculate_fine(self):
        if self.days_late<=5:
            fine = self.days_late * 5
        elif self.days_late>5 and self.days_late<=10:
            fine = (5 * 5) + (self.days_late - 5) * 10
        elif self.days_late>10:
            fine = (5 * 5) + (5 * 10) + (self.days_late - 10) * 20
        return fine
    def display_book_details(self):
        fine = self.calculate_fine()
        print(f"Book ID: {self.book_id}")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Barrower: {self.barrower}")
        print(f"Days Late: {self.days_late}")
        print(f"Total Fine: {fine}")
# Example usage:
book1 = LibraryBook(401, "1984", "George Orwell", "Eve Adams", 7)
book1.display_book_details()
book2 = LibraryBook(402, "To Kill a Mockingbird", "Harper Lee", "Frank Miller", 12)
book2.display_book_details()
```

Code:

```python
class LibraryBook:
    def __init__(self,book_id,title,author,barrower,days_late):
```

```python
        self.book_id = book_id
        self.title = title
        self.author = author
        self.barrower = barrower
        self.days_late = days_late
    def calculate_fine(self):
        if self.days_late<=5:
            fine = self.days_late * 5
        elif self.days_late>5 and self.days_late<=10:
            fine = (5 * 5) + (self.days_late - 5) * 10
        elif self.days_late>10:
            fine = (5 * 5) + (5 * 10) + (self.days_late - 10) * 20
        return fine
    def display_book_details(self):
        fine = self.calculate_fine()
        print(f"Book ID: {self.book_id}")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Barrower: {self.barrower}")
        print(f"Days Late: {self.days_late}")
        print(f"Total Fine: {fine}")
# Example usage:
book1 = LibraryBook(401, "1984", "George Orwell", "Eve Adams", 7)
book1.display_book_details()
book2 = LibraryBook(402, "To Kill a Mockingbird", "Harper Lee", "Frank
Miller", 12)
book2.display_book_details()
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/One
Drive/Pictures/Desktop/AIAC/lab_05 (1).py"
Book ID: 401
Title: 1984
Author: George Orwell
Barrower: Eve Adams
Days Late: 7
Total Fine: 45
Book ID: 402
Title: To Kill a Mockingbird
Author: Harper Lee
Barrower: Frank Miller
Days Late: 12
Total Fine: 115
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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.

Screenshots:

```python
def student_report(student_data):
    report = {}
    for student in student_data:
        name = student['name']
        scores = student['scores']
        total_score = sum(scores)
        average_score = total_score / len(scores) if scores else 0
        report[name] = {
            'Total Score': total_score,
            'Average Score': average_score
        }
    return report
```

```python
166
167  def student_report(student_data):
168      report = {}
169      for student in student_data:
170          name = student['name']
171          scores = student['scores']
172          total_score = sum(scores)
173          average_score = total_score / len(scores) if scores else 0
174          pass_count = len([score for score in scores if score >= 40])
175          report[name] = {
176              'average_score': average_score,
177              'pass_count': pass_count
178          }
179      return report
180  # Example usage:
181  students = [
182      {'name': 'Alice', 'scores': [85, 42, 39, 76]},
183      {'name': 'Bob', 'scores': [90, 55, 28, 67]},
184      {'name': 'Charlie', 'scores': [49, 100, 38, 45]}
185  ]
186  report = student_report(students)
187  for name, details in report.items():
188      print(f"Student: {name}, Average Score: {details['average_score']}, Subjects Passed: {details['pass_count']}")
189
190
```

Code:

```python
def student_report(student_data):
    report = {}
    for student in student_data:
        name = student['name']
```

```python
        scores = student['scores']
        total_score = sum(scores)
        average_score = total_score / len(scores) if scores else 0
        pass_count = len([score for score in scores if score >= 40])
        report[name] = {
            'average_score': average_score,
            'pass_count': pass_count
        }
    return report
# Example usage:
students = [
    {'name': 'Alice', 'scores': [85, 42, 39, 76]},
    {'name': 'Bob', 'scores': [90, 55, 28, 67]},
    {'name': 'Charlie', 'scores': [49, 100, 38, 45]}
]
report = student_report(students)
for name, details in report.items():
    print(f"Student: {name}, Average Score: {details['average_score']},
Subjects Passed: {details['pass_count']}")
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/One
Drive/Pictures/Desktop/AIAC/lab_05 (1).py"
Student: Alice, Average Score: 60.5, Subjects Passed: 3
Student: Bob, Average Score: 60.0, Subjects Passed: 3
Student: Charlie, Average Score: 58.0, Subjects Passed: 3
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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.

Screenshots:

```
116
117    class TaxiRide:
118        def __init__(self,ride_id,driver_name,distance_traveled,waiting_time_minutes):
119            self.ride_id = ride_id
120            self.driver_name = driver_name
121            self.distance_traveled = distance_traveled
122            self.waiting_time_minutes = waiting_time_minutes
123        def calculate_fare(self):
124            if self.distance_traveled <= 10:
125                fare = self.distance_traveled * 15
126            elif self.distance_traveled > 10 and self.distance_traveled <= 30:
127                fare = (10 * 15) + (self.distance_traveled - 10) * 12
128            elif self.distance_traveled >30:
129                fare = (10 * 15) + (40 * 12) + (self.distance_traveled - 50) * 10
130            waiting_charge = self.waiting_time_minutes * 2
131            total_fare = fare + waiting_charge
132            return total_fare
133        def display_ride_details(self):
               total_fare = self.calculate_fare()
               print(f"Ride ID: {self.ride_id}")
               print(f"Driver Name: {self.driver_name}")
               print(f"Distance Traveled: {self.distance_traveled} km")
               print(f"Waiting Time: {self.waiting_time_minutes} minutes")
               print(f"Total Fare: {total_fare}")
134
135        ^
```

```
116
117    class TaxiRide:
118        def __init__(self,ride_id,driver_name,distance_traveled,waiting_time_minutes):
119            self.ride_id = ride_id
120            self.driver_name = driver_name
121            self.distance_traveled = distance_traveled
122            self.waiting_time_minutes = waiting_time_minutes
123        def calculate_fare(self):
124            if self.distance_traveled <= 10:
125                fare = self.distance_traveled * 15
126            elif self.distance_traveled > 10 and self.distance_traveled <= 30:
127                fare = (10 * 15) + (self.distance_traveled - 10) * 12
128            elif self.distance_traveled >30:
129                fare = (10 * 15) + (40 * 12) + (self.distance_traveled - 50) * 10
130            waiting_charge = self.waiting_time_minutes * 2
131            total_fare = fare + waiting_charge
132            return total_fare
133        def display_ride_details(self):
134            total_fare = self.calculate_fare()
135            print(f"Ride ID: {self.ride_id}")
136            print(f"Driver Name: {self.driver_name}")
137            print(f"Distance Traveled: {self.distance_traveled} km")
138            print(f"Waiting Time: {self.waiting_time_minutes} minutes")
139            print(f"Total Fare: {total_fare}")
140    # Example usage:
141    ride1 = TaxiRide(501, "George Harris", 25, 10)
142    ride1.display_ride_details()
143    ride2 = TaxiRide(502, "Hannah Lee", 55, 5)
144    ride2.display_ride_details()
145
```

Code:

```python
class TaxiRide:
    def
__init__(self,ride_id,driver_name,distance_traveled,waiting_time_minutes):
```

```python
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_traveled = distance_traveled
        self.waiting_time_minutes = waiting_time_minutes
    def calculate_fare(self):
        if self.distance_traveled <= 10:
            fare = self.distance_traveled * 15
        elif self.distance_traveled > 10 and self.distance_traveled <= 30:
            fare = (10 * 15) + (self.distance_traveled - 10) * 12
        elif self.distance_traveled >30:
            fare = (10 * 15) + (40 * 12) + (self.distance_traveled - 50) * 10
        waiting_charge = self.waiting_time_minutes * 2
        total_fare = fare + waiting_charge
        return total_fare
    def display_ride_details(self):
        total_fare = self.calculate_fare()
        print(f"Ride ID: {self.ride_id}")
        print(f"Driver Name: {self.driver_name}")
        print(f"Distance Traveled: {self.distance_traveled} km")
        print(f"Waiting Time: {self.waiting_time_minutes} minutes")
        print(f"Total Fare: {total_fare}")
# Example usage:
ride1 = TaxiRide(501, "George Harris", 25, 10)
ride1.display_ride_details()
ride2 = TaxiRide(502, "Hannah Lee", 55, 5)
ride2.display_ride_details()
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/One
Drive/Pictures/Desktop/AIAC/lab_05 (1).py"
Ride ID: 501
Driver Name: George Harris
Distance Traveled: 25 km
Waiting Time: 10 minutes
Total Fare: 350
Ride ID: 502
Driver Name: Hannah Lee
Distance Traveled: 55 km
Waiting Time: 5 minutes
Total Fare: 690
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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

Screenshots:

```
147
148 v def statistics_subjects(scrores_list):
149 v     if not scrores_list:
150             return None, None, None
151         lowest_score = min(scrores_list)
152         highest_score = max(scrores_list)
153         average_score = sum(scrores_list) / len(scrores_list)
154         numberofstudentspassed = len([score for score in scrores_list if score >= 40])
155         numberoftsudentsfailed = len([score for score in scrores_list if score < 40])
156         print(f"Number of Students Passed: {numberofstudentspassed}")
157         print(f"Number of Students Failed: {numberoftsudentsfailed}")
158         return lowest_score, highest_score, average_score
159     # Example usage:
160
```

```
147
148     def statistics_subjects(scrores_list):
149         if not scrores_list:
150             return None, None, None
151         lowest_score = min(scrores_list)
152         highest_score = max(scrores_list)
153         average_score = sum(scrores_list) / len(scrores_list)
154         numberofstudentspassed = len([score for score in scrores_list if score >= 40])
155         numberoftsudentsfailed = len([score for score in scrores_list if score < 40])
156         print(f"Number of Students Passed: {numberofstudentspassed}")
157         print(f"Number of Students Failed: {numberoftsudentsfailed}")
158         return lowest_score, highest_score, average_score
159
160     # Example usage:
161     scores = [85, 42, 39, 76, 90, 55, 28, 67, 49, 100]
162     lowest, highest, average = statistics_subjects(scores)
163     print(f"Lowest Score: {lowest}")
164     print(f"Highest Score: {highest}")
165     print(f"Average Score: {average}")
166
```

Code:

```
def statistics_subjects(scrores_list):
    if not scrores_list:
        return None, None, None
```

```
        lowest_score = min(scrores_list)
        highest_score = max(scrores_list)
        average_score = sum(scrores_list) / len(scrores_list)
        numberofstudentspassed = len([score for score in scrores_list if score >=
40])
        numberoftsudentsfailed = len([score for score in scrores_list if score <
40])
        print(f"Number of Students Passed: {numberofstudentspassed}")
        print(f"Number of Students Failed: {numberoftsudentsfailed}")
        return lowest_score, highest_score, average_score


# Example usage:
scores = [85, 42, 39, 76, 90, 55, 28, 67, 49, 100]
lowest, highest, average = statistics_subjects(scores)
print(f"Lowest Score: {lowest}")
print(f"Highest Score: {highest}")
print(f"Average Score: {average}")
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/One
Drive/Pictures/Desktop/AIAC/lab_05 (1).py"
Number of Students Passed: 8
Number of Students Failed: 2
Lowest Score: 28
Highest Score: 100
Average Score: 63.1
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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.

Prompt:

#write a python code to print prime or not using basic method in an basic level of code without any function in native approach

Screenshots:

```
#write a python code to print prime or not using basic method in an basic level of code without any function in native approach
num = int(input("Enter a number: "))
if num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            print(f"{num} is not a prime number")
            break
    else:
        print(f"{num} is a prime number")
```

```
#write a python code to print prime or not using basic method in an
num = int(input("Enter a number: "))
if num > 1:
    for i in range(2, int(num**0.5) + 1):
        if (num % i) == 0:
            print(f"{num} is not a prime number")
            break
    else:
        print(f"{num} is a prime number")
else:
    print(f"{num} is not a prime number")
```

Code:

#write a python code to print prime or not using basic method in an basic level of code without any function in native approach

```
num = int(input("Enter a number: "))
if num > 1:
    for i in range(2, int(num**0.5) + 1):
        if (num % i) == 0:
            print(f"{num} is not a prime number")
            break
    else:
        print(f"{num} is a prime number")
else:
    print(f"{num} is not a prime number")
```

optimized code :

prompt:

#write a python code to print weather the gievn number is prime or not in function method and in optimized way

```
#write a python code to print weather the gievn number is prime or not in function method and in optimized way
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True
```

```
201
202    #write a python code to print weather the gievn number is prime or not in function method and in optimized way
203    def is_prime(num):
204        if num <= 1:
205            return False
206        for i in range(2, int(num**0.5) + 1):
207            if (num % i) == 0:
208                return False
209        return True
210    # Example usage:
211    number = int(input("Enter a number: "))
212    if is_prime(number):
213        print(f"{number} is a prime number")
214    else:
215        print(f"{number} is not a prime number")
216
217    |
```

Code:

#write a python code to print weather the gievn number is prime or not in function method and in optimized way

```python
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num**0.5) + 1):
        if (num % i) == 0:
            return False
    return True
# Example usage:
number = int(input("Enter a number: "))
if is_prime(number):
    print(f"{number} is a prime number")
else:
    print(f"{number} is not a prime number")
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/
Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/lab_05 (1).py"
Enter a number: 19
19 is a prime number
Enter a number:
```

Explaination :

The first method uses a basic native approach without functions, where it directly checks if a number is prime by iterating from 2 to the square root of the number and breaking out of the loop immediately upon finding a divisor, printing the result inline within the control flow structure. The second method is an optimized function-based approach that encapsulates the prime-checking logic into a reusable is_prime() function that returns a boolean value (True or False) for the given number; while both methods use the same efficient algorithm of checking only up to the square root of the number, the function-based approach is cleaner, more modular, and easier to reuse across multiple calls, whereas the first approach directly ties the logic to I/O operations and makes it difficult to reuse the logic without code duplication.

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:

#write a recursive function to calculate Fibonacci series up to n terms and explain the code and recurision in comments

Screenshots:

```python
#write a recursive function to calculate Fibonacci series up to n terms and explain the code and recur
def fibonacci(n):
    # Base case: if n is 0, return an empty list
    if n <= 0:
        return []
    # Base case: if n is 1, return a list with the first Fibonacci number
    elif n == 1:
        return [0]
    # Base case: if n is 2, return a list with the first two Fibonacci numbers
    elif n == 2:
        return [0, 1]
    else:
        # Recursive case: get the Fibonacci series up to (n-1) terms
        fib_series = fibonacci(n - 1)
        # Calculate the next Fibonacci number by summing the last two numbers in the series
        next_fib = fib_series[-1] + fib_series[-2]
        # Append the next Fibonacci number to the series
        fib_series.append(next_fib)
        return fib_series
```

```python
216
217   #write a recursive function to calculate Fibonacci series up to n terms and explain the code and recurisi
218   def fibonacci(n):
219       # Base case: if n is 0, return an empty list
220       if n <= 0:
221           return []
222       # Base case: if n is 1, return a list with the first Fibonacci number
223       elif n == 1:
224           return [0]
225       # Base case: if n is 2, return a list with the first two Fibonacci numbers
226       elif n == 2:
227           return [0, 1]
228       else:
229           # Recursive case: get the Fibonacci series up to (n-1) terms
230           fib_series = fibonacci(n - 1)
231           # Calculate the next Fibonacci number by summing the last two numbers in the series
232           next_fib = fib_series[-1] + fib_series[-2]
233           # Append the next Fibonacci number to the series
234           fib_series.append(next_fib)
235           return fib_series
236   # Example usage:
237   terms = int(input("Enter the number of terms for Fibonacci series: "))
238   fib_sequence = fibonacci(terms)
239   print(f"Fibonacci series up to {terms} terms: {fib_sequence}")
240
241
242
```

Code:

#write a recursive function to calculate Fibonacci series up to n terms and explain the code and recurision in comments

```python
def fibonacci(n):
    # Base case: if n is 0, return an empty list
    if n <= 0:
        return []
    # Base case: if n is 1, return a list with the first Fibonacci number
    elif n == 1:
```

```
            return [0]
    # Base case: if n is 2, return a list with the first two Fibonacci
numbers
    elif n == 2:
        return [0, 1]
    else:
        # Recursive case: get the Fibonacci series up to (n-1) terms
        fib_series = fibonacci(n - 1)
        # Calculate the next Fibonacci number by summing the last two numbers
in the series
        next_fib = fib_series[-1] + fib_series[-2]
        # Append the next Fibonacci number to the series
        fib_series.append(next_fib)
        return fib_series
# Example usage:
terms = int(input("Enter the number of terms for Fibonacci series: "))
fib_sequence = fibonacci(terms)
print(f"Fibonacci series up to {terms} terms: {fib_sequence}")
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/
Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/lab_05 (1).py"
Enter the number of terms for Fibonacci series: 5
Fibonacci series up to 5 terms: [0, 1, 1, 2, 3]
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

Explanation:

Comments explain the why and intent behind code, not just the what, so future readers (including you) can understand decisions, assumptions, and tricky logic quickly without re-deriving it. They make maintenance, debugging, and collaboration easier by clarifying purpose, edge cases, and expected behavior, especially when the code is complex or not self-explanatory.

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:

#write a python program that reads a text file and processes the data Generate code with proper error handling and clear explanations for each exception

Screenshots:

```
#write a python program that reads a text file and processes the data Generate code with proper er
try:
    # Attempt to open the file in read mode
    with open('data.txt', 'r') as file:
        try:
            # Read the contents of the file
            data = file.read()
            # Process the data (for example, count the number of lines)
            lines = data.splitlines()
            print(f"The file contains {len(lines)} lines.")
        except Exception as e:
            # Handle any exceptions that occur during data processing
            print(f"An error occurred while processing the data: {e}")
```

```
#write a python program that reads a text file and processes the data Generate code with proper error han
try:
    # Attempt to open the file in read mode
    with open('data.txt', 'r') as file:
        try:
            # Read the contents of the file
            data = file.read()
            # Process the data (for example, count the number of lines)
            lines = data.splitlines()
            print(f"The file contains {len(lines)} lines.")
        except Exception as e:
            # Handle any exceptions that occur during data processing
            print(f"An error occurred while processing the data: {e}")
except FileNotFoundError:
    # Handle the case where the file does not exist
    print("Error: The file 'data.txt' was not found.")
except IOError:
    # Handle other I/O errors
    print("Error: An I/O error occurred while trying to read the file.")
except Exception as e:
    # Handle any other unexpected exceptions
    print(f"An unexpected error occurred: {e}")
```

Code:

#write a python program that reads a text file and processes the data Generate code with proper error handling and clear explanations for each exception

```python
try:
    # Attempt to open the file in read mode
    with open('data.txt', 'r') as file:
        try:
            # Read the contents of the file
            data = file.read()
            # Process the data (for example, count the number of lines)
            lines = data.splitlines()
            print(f"The file contains {len(lines)} lines.")
        except Exception as e:
            # Handle any exceptions that occur during data processing
            print(f"An error occurred while processing the data: {e}")
except FileNotFoundError:
    # Handle the case where the file does not exist
    print("Error: The file 'data.txt' was not found.")
except IOError:
    # Handle other I/O errors
    print("Error: An I/O error occurred while trying to read the file.")
except Exception as e:
    # Handle any other unexpected exceptions
    print(f"An unexpected error occurred: {e}")
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/
Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/lab_05 (1).py"
The file contains 1 lines.
```

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/
Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/lab_05 (1).py"
Error: The file 'data.txt' was not found.
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> []
```

Explanation:

That error-handling block tries to open data.txt and read it safely: the outer try opens the file, and if it's missing it catches FileNotFoundError, while other I/O problems are caught by IOError. Inside, a nested try handles problems that might occur while processing the file content (like splitting or counting), and any unexpected issues fall to the final generic Exception handler, so the program fails gracefully with clear messages instead of crashing.