**Home Assignment – I**

|  |
| --- |
| **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. |
| **Task 2: Electricity Bill Calculation** Create Python code that defines a class named ElectricityBill with attributes: customer\_id, name, 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. |
| **Task 3: Product Discount Calculation** Create Python code that defines a class named Product with attributes: product\_id, product\_name, price, 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. |
| **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. |
| **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. |
| **Task 6: Taxi Fare Calculation** Create Python code that defines a class named TaxiRide with attributes: ride\_id, driver\_name, distance\_km, 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. |
| **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.  **Your function should return the following:**   1. **Highest score** in the class 2. **Lowest score** in the class 3. **Class average** score 4. **Number of students passed** (pass criteria: score ≥ 40) 5. **Number of students failed** (score < 40)   Allow Copilot to assist with aggregations and logic |
| **Task 8: Movie Ticket Booking** Create Python code that defines a class named MovieTicket with attributes: ticket\_id, movie\_name, seat\_type, no\_of\_tickets. Implement a method display\_details() to print ticket details, and a method calculate\_price() where:   * Normal seat → ₹150 per ticket * Premium seat → ₹250 per ticket * VIP seat → ₹400 per ticket Create a ticket object, display details, and print the total cost. |
| **Task 9: Parking Fee Calculation** Create Python code that defines a class named ParkingLot with attributes: vehicle\_no, vehicle\_type, hours\_parked. Implement a method display\_details() to print parking details, and a method calculate\_fee() where:   * Two-wheeler → ₹10 per hour * Car → ₹20 per hour * Truck → ₹30 per hour Create a parking object, display details, and print the total fee. |
| **Task 10: Mobile Data Plan Usage** Create Python code that defines a class named MobilePlan with attributes: plan\_id, customer\_name, data\_used\_gb, base\_charge. Implement a method display\_details() to print plan details. Implement another method calculate\_extra\_charge() where:   * Data used ≤ 10 GB → No extra charge * 11 to 20 GB → ₹50 per extra GB * More than 20 GB → ₹75 per extra GB Create at least one plan object, display details, and print the total charge |

**Instructions to submit the document**

* 1. **Analyze the given task and write relevant prompt**
  2. **Analyze the suggestions generated by AI tool**
  3. **Run the code**
  4. **If desired output generated make a note or refine the prompt to generate output,**
  5. **take screenshot of generated code and output, then paste it in solution file**
  6. **upload solution file (hallticket1.doc / pdf format only) into the shared google drive link**

**file format : 2403A7001.doc or 2403A7001.pdf**

* 1. **Google form link:**
  2. **Complete the task Submit within deadline on or before 17-08-2025 1:00pm(AN)**
  3. **Sample template enclosed here**

|  |
| --- |
| **Sample solution template**  **Task 0: Car Insurance Premium Calculation Prompt:**  Create Python code that defines a class named CarInsurance with attributes: policy\_no, owner\_name, car\_model, car\_age, base\_premium. Implement a method display\_details() to print all policy details. Implement another method calculate\_premium() to add extra charges based on car age:   * Car age ≤ 3 years → Extra 5% of base\_premium * 4 to 7 years → Extra 10% of base\_premium * More than 7 years → Extra 15% of base\_premium Create a policy object, display details, and print the total premium.   **Code :**  class CarInsurance:      def \_\_init\_\_(self, policy\_no, owner\_name, car\_model, car\_age, base\_premium):          self.policy\_no = policy\_no          self.owner\_name = owner\_name          self.car\_model = car\_model          self.car\_age = car\_age          self.base\_premium = base\_premium      def display\_details(self):          print(f"Policy No: {self.policy\_no}")          print(f"Owner Name: {self.owner\_name}")          print(f"Car Model: {self.car\_model}")          print(f"Car Age: {self.car\_age} years")          print(f"Base Premium: ${self.base\_premium}")      def calculate\_premium(self):          if self.car\_age <= 3:              extra\_charge = 0.05 \* self.base\_premium          elif 4 <= self.car\_age <= 7:              extra\_charge = 0.10 \* self.base\_premium          else:              extra\_charge = 0.15 \* self.base\_premium          total\_premium = self.base\_premium + extra\_charge          print(f"Total Premium (with extra charges): ${total\_premium}")  # Create a policy object  policy = CarInsurance("POL12345", "John Doe", "Toyota Camry", 5, 500)  # Display policy details  policy.display\_details()  # Calculate and print the total premium  policy.calculate\_premium()  **Output :**  **Remarks:** The solution demonstrates a solid understanding of object-oriented programming by encapsulating policy details and premium calculation within the CarInsurance class. Attribute initialization is handled cleanly through the constructor, and methods are well-separated for clarity—display\_details() for information output and calculate\_premium() for logic processing. The premium calculation correctly applies age-based percentage increments, ensuring accuracy in results. Variable naming is descriptive, enhancing code readability, and the formatted print statements make the output easy to understand for end users. Overall, the implementation is both functional and well-organized. |