ASSIGNMENT - 5.1

KONDAPALLI DEVI PRIYANKA
2303A51108

Task 1:
Employee Data: Create Python code that defines a class named`Employee 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,exp=0):
        self.empid = empid
        self.empname = empname
        self.designation = designation
        self.basic_salary = basic_salary
        self.exp = exp
        def display_details(self):
            print(f"Employee ID: {self.empid}")
            print(f"Employee Name: {self.empname}")
            print(f"Designation: {self.designation}")
            print(f"Basic Salary: {self.basic_salary}")
            print(f"Experience: {self.exp} years")
        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
            elif self.exp < 5 :
                allowance = 0.05 * self.basic_salary
            return allowance
```

```
emp1 = employee(101, "Alice", "Manager", 80000, 12)
emp2 = employee(102, "Bob", "Developer", 60000, 7)
emp3 = employee(103, "Charlie", "Intern", 30000, 2)
emp1.display_details()
emp2.display_details()
emp3.display_details()

employee.display_details(emp1)
employee.calculate_allowance(emp1)

employee.display_details(emp2)
employee.calculate_allowance(emp2)
```

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 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) + (100 * 7) + (self.units_consumed -
200) * 10
        return bill_amount
```

```
    def display_bill(self):
        bill_amount = self.calculate_bill()
        print(f"Customer Name: {self.customer_name}")
        print(f"Units Consumed: {self.units_consumed}")
        print(f"Total Bill Amount: ${bill_amount:.2f}")
    def display_details(self):
        print(f"Customer ID: {self.customer_id}")
        print(f"Customer Name: {self.customer_name}")
        print(f"Units Consumed: {self.units_consumed}")
customer1 = electricitybill(1, "John Doe", 250)
customer2 = electricitybill(2, "Jane Smith", 350)
total_bill1 = customer1.calculate_bill()
total_bill2 = customer2.calculate_bill()
customer1.display_bill()
customer2.display_bill()

customer1.display_details()
customer2.display_details()
```

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.

```
class product:
    def __init__(self, product_name, product_id, price, category,
quantity):
        self.id = product_id
        self.name = product_name
        self.price = price
        self.category = category
        self.quantity = quantity
```

```
    def calulate_discount(self):
        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
        return discount
    def display_details(self):
        print(f"Product ID: {self.id}")
        print(f"Product Name: {self.name}")
        print(f"Category: {self.category}")
        print(f"Price: ${self.price:.2f}")
        print(f"Quantity: {self.quantity}")

    def total_cost(self):
        return self.price * self.quantity
prod1 = product("Laptop", 201, 1200.00, "Electronics", 2)
prod2 = product("Jeans", 202, 50.00, "Clothing", 3)
prod3 = product("Apples", 203, 2.00, "Groceries", 10)
prod1.display_details()
prod2.display_details()
prod3.display_details()
print(f"Discount on {prod1.name}: ${prod1.calulate_discount():.2f}")
print(f"Discount on {prod2.name}: ${prod2.calulate_discount():.2f}")
print(f"Discount on {prod3.name}: ${prod3.calulate_discount():.2f}")
print(f"Total cost for {prod1.name}: ${prod1.total_cost():.2f}")
print(f"Total cost for {prod2.name}: ${prod2.total_cost():.2f}")
print(f"Total cost for {prod3.name}: ${prod3.total_cost():.2f}")
product.display_details(prod1)
product.calulate_discount(prod1)
product.display_details(prod2)
product.calulate_discount(prod2)
product.display_details(prod3)
product.calulate_discount(prod3)
```

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.

```
class libraryBook:
    def __init__(self, book_id, title, author, bowrower, days_late):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.bowrower = bowrower
        self.days_late = days_late

    def calculate_fine(self):
        if self.days_late <= 0:
            fine = 0
        elif self.days_late <= 5:
            fine = self.days_late * 5.00
        elif self.days_late <= 10:
            fine = (5 * 5.00) + (self.days_late - 5) * 10.00
        else:
            fine = (5 * 5.00) + (5 * 10.00) + (self.days_late - 10) *
15.00
        return fine
    def display_details(self):
        print(f"Book ID: {self.book_id}")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Bowrower: {self.bowrower}")
        print(f"Days Late: {self.days_late}")
book1 = libraryBook(301, "The Great Gatsby", "F. Scott Fitzgerald",
"David", 3)
book2 = libraryBook(302, "1984", "George Orwell", "Emma", 7)
book3 = libraryBook(303, "To Kill a Mockingbird", "Harper Lee", "Olivia",
12)
book1.display_details()
```

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.

```
def student_report(student_data):
    report = []

    for name, marks in student_data.items():
        average = sum(marks) / len(marks)
        status = "Pass" if average >= 40 else "Fail"

        report.append({
            "name": name,
            "average": average,
            "status": status
        })

    return report


student_data = {
    "Ravi": [45, 50, 55],
    "Anu": [30, 35, 40],
    "Kiran": [70, 65, 80]
}

result = student_report(student_data)

for student in result:
    print(student)
```

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.

```
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 calculate_fare(self):
        base_fare = 50  # Base fare in currency units
        cost_per_km = 20  # Cost per kilometer
        cost_per_minute_waiting = 2  # Cost per minute of waiting

        fare = base_fare + (self.distance_km * cost_per_km) +
(self.waiting_time_min * cost_per_minute_waiting)
        return fare
    def display_details(self):
        print(f"Ride ID: {self.ride_id}")
        print(f"Driver Name: {self.driver_name}")
        print(f"Distance (km): {self.distance_km}")
        print(f"Waiting Time (min): {self.waiting_time_min}")
ride1 = TaxiRide(401, "John", 15, 5)
ride2 = TaxiRide(402, "Alice", 8, 10)
ride1.display_details()
ride2.display_details()
print(f"Fare for Ride 1: ${ride1.calculate_fare():.2f}")
print(f"Fare for Ride 2: ${ride2.calculate_fare():.2f}")
TaxiRide.display_details(ride1)
```

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