

Name: Muhammad Abdullah

Roll no: 016

Section: 3A

Department: BSDSM

Task #3: Artificial Intelligence Lab

Q1. Create a payroll system using static variables to track total employees and total salary expense. Every time a new employee is added, both values must update automatically, and a static method should display the company payroll summary.

```
main.py Company Payroll Summary
          Total Employees: 3
          Total Salary Expense: 165000
          === Code Execution Successful ===

1- class Employee:
2     total_employees = 0
3     total_salary_expense = 0
4
5     def __init__(self, name, salary):
6         self.name = name
7         self.salary = salary
8         Employee.total_employees += 1
9         Employee.total_salary_expense += salary
10
11    @staticmethod
12    def payroll_summary():
13        print("Company Payroll Summary")
14        print("Total Employees:", Employee.total_employees)
15        print("Total Salary Expense:", Employee.total_salary_expense)
16
17 e1 = Employee("Rasikh", 50000)
18 e2 = Employee("Ali", 60000)
19 e3 = Employee("Jinnah", 55000)
20 Employee.payroll_summary()
```

Q2. Design a library system where static variables track total books and issued books. Implement issue and return logic so that counters remain consistent and incorrect issuing is prevented.

```

main.py | Run | Output | Clear
1  class LibraryBook:
2      total_books = 0
3      issued_books = 0
4      def __init__(self, title):
5          self.title = title
6          self.is_issued = False
7          LibraryBook.total_books += 1
8      def issue_book(self):
9          if not self.is_issued:
10             self.is_issued = True
11             LibraryBook.issued_books += 1
12             print(f"{self.title} has been issued.")
13         else:
14             print(f"{self.title} is already issued. Cannot issue again.")
15     def return_book(self):
16         if self.is_issued:
17             self.is_issued = False
18             LibraryBook.issued_books -= 1
19             print(f"{self.title} has been returned.")
20         else:
21             print(f"{self.title} was not issued.")
22     @staticmethod
23     def library_summary():
24         print("----- Library Summary -----")
25         print(f"Total Books: {LibraryBook.total_books}")
26         print(f"Issued Books: {LibraryBook.issued_books}")
27         print(f"Available Books: {LibraryBook.total_books - LibraryBook.issued_books}")
28 b1 = LibraryBook("Python")
29 b2 = LibraryBook("Data Science")
30 b3 = LibraryBook("AI Basics")
31 b1.issue_book()
32 b1.issue_book()
33 b2.issue_book()
34 b1.return_book()
35 LibraryBook.library_summary()

```

Output:

```

Python has been issued.
Python is already issued. Cannot issue again.
Data Science has been issued.
Python has been returned.
----- Library Summary -----
Total Books: 3
Issued Books: 1
Available Books: 2
== Code Execution Successful ==

```

Q3. Build an employee performance system where static data maintains total employees and average performance score, updating automatically whenever a new employee is added.

```

main.py | Run | Output | Clear
1  class Employee:
2      total_employees = 0
3      total_score = 0
4      def __init__(self, name, performance_score):
5          self.name = name
6          self.performance_score = performance_score
7          Employee.total_employees += 1
8          Employee.total_score += performance_score
9      @staticmethod
10     def performance_summary():
11         if Employee.total_employees == 0:
12             average = 0
13         else:
14             average = Employee.total_score / Employee.total_employees
15         print(" Performance Summary ")
16         print("Total Employees: ", Employee.total_employees)
17         print("Average Performance Score: ", round(average, 2))
18 e1 = Employee("Rasikh", 85)
19 e2 = Employee("Sana", 90)
20 e3 = Employee("Ghandi", 75)
21 Employee.performance_summary()

```

Output:

```

Performance Summary
Total Employees: 3
Average Performance Score: 83.33
== Code Execution Successful ==

```

Q4. Create a class that uses a loop to process student attendance and static variables to track total present and absent students. Use if/else logic and a static method to update and display results.

```

main.py | Run | Output
1 - class Attendance:
2     total_present = 0
3     total_absent = 0
4 - def __init__(self, name):
5     self.name = name
6     self.is_present = False
7
8 - def mark_attendance(self, status):
9     if status.lower() == "p":
10        self.is_present = True
11        Attendance.total_present += 1
12    else:
13        self.is_present = False
14        Attendance.total_absent += 1
15    @staticmethod
16    def attendance_summary():
17        print("Attendance Summary:")
18        print(f"Total Present Students: {Attendance.total_present}")
19        print(f"Total Absent Students: {Attendance.total_absent}")
20    students = ["Rasikh", "Sania", "Jeffery", "Donald", "Gates"]
21    for student_name in students:
22        student = Attendance(student_name)
23        status = input(f"Is {student_name} present or absent? ")
24        student.mark_attendance(status)
25    Attendance.attendance_summary()

```

Is Rasikh present or absent? p
Is Sania present or absent? p
Is Jeffery present or absent? p
Is Donald present or absent? a
Is Gates present or absent? a
Attendance Summary:
Total Present Students: 3
Total Absent Students: 2
== Code Execution Successful ==

Q5. Design an inventory system where static variables track total items and sold items. Use loops to process sales data and if/else to prevent over-selling. Include a class method to reset inventory data

```

main.py | Run | Output | Clear
1 - class InventoryItem:
2     total_items = 0
3     sold_items = 0
4 - def __init__(self, name, quantity):
5     self.name = name
6     self.quantity = quantity
7     InventoryItem.total_items += quantity
8 - def sell(self, quantity):
9     if quantity <= self.quantity:
10        self.quantity -= quantity
11        InventoryItem.sold_items += quantity
12        print(f"{quantity} {self.name}(s) sold.")
13    else:
14        print(f"Cannot sell {quantity} {self.name}(s). Only {self.quantity} available.")
15    @classmethod
16    def reset_inventory(cls):
17        cls.total_items = 0
18        cls.sold_items = 0
19        print("Inventory data has been reset.")
20    @staticmethod
21    def inventory_summary():
22        print("----- Inventory Summary -----")
23        print(f"Total Items in Stock: {InventoryItem.total_items - InventoryItem.sold_items}")
24        print(f"Total Sold Items: {InventoryItem.sold_items}")
25    item1 = InventoryItem("Laptop", 10)
26    item2 = InventoryItem("Phone", 20)
27    sales_data = [
28        ("Laptop", 3),
29        ("Phone", 5),
30        ("Laptop", 8),
31        ("Phone", 10)
32    ]
33    for item_name, qty in sales_data:
34        if item_name == "Laptop":
35            item1.sell(qty)
36        elif item_name == "Phone":
37            item2.sell(qty)
38    InventoryItem.inventory_summary()
39    InventoryItem.reset_inventory()
40    InventoryItem.inventory_summary()

```

3 Laptop(s) sold.
5 Phone(s) sold.
Cannot sell 8 Laptop(s). Only 7 available.
10 Phone(s) sold.
----- Inventory Summary -----
Total Items in Stock: 12
Total Sold Items: 18
Inventory data has been reset.
----- Inventory Summary -----
Total Items in Stock: 0
Total Sold Items: 0
== Code Execution Successful ==