SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName:M. Tech		**AssignmentType: Lab**	**AcademicYear:**2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
CourseCode		**CourseTitle**	AI Assisted Problem Solving Using Python
Year/Sem	II/I	**Regulation**	R24
DateandDay of Assignment	Week5 - Monday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:10.1(Presentassignmentnumber)/**24**(Totalnumberofassignments)			

Q.No.	Question	*ExpectedTime to complete*

1	**Lab 10 – Code Review and Quality: Using AI to Improve Code Quality and Readability**
	Lab Objectives
	• Use AI for automated code review and quality enhancement.
	• Identify and fix syntax, logical, performance, and security issues in Python code.
	• Improve readability and maintainability through structured refactoring and comments.
	• Apply prompt engineering for targeted improvements.
	• Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices
	--- 	
	Task Description #1 – Refactor Nested Conditionals
	Task: Provide AI with the following nested conditional code and ask it to simplify and refactor for readability.
	
	Python script.
	def discount(price, category):
	  if category == "student":
	  if price > 1000:
	  return price * 0.9
	  else:
	  return price * 0.95
	  else:
	  if price > 2000:
	  return price * 0.85
		Week5 - Monday

```
    else:
        return price
```

Expected Output:

- Refactored code using cleaner logic, possibly a dictionary or separate helper functions.

Task Description #2 – Optimize Redundant Loops

Task: Give AI this messy loop and ask it to refactor and optimize.

Python script

```
def find_common(a, b):
    res = []
    for i in a:
        for j in b:
            if i == j:
                res.append(i)
    return res
```

Expected Output:

Cleaner version using Python sets (set(a) & set(b))

Task Description #3 – Improve Class Design

Task: Provide this class with poor readability and ask AI to improve:

- Naming conventions
- Encapsulation
- Readability & maintainability

Python Script

```
class emp:
    def __init__(self,n,s):
        self.n=n
        self.s=s
    def inc(self,p):
        self.s=self.s+(self.s*p/100)
    def pr(self):
        print("emp:",self.n,"salary:",self.s)
```

Expected Output:

- Employee class with meaningful methods (increase_salary, display_info), formatted output, and added docstrings.

Task Description #4 – Modularize Long Function

Task: Give AI this long unstructured function and let it modularize into smaller helper functions.

Python Script

```
def process_scores(scores):
    total = 0
    for s in scores:
        total += s
    avg = total / len(scores)

    highest = scores[0]
    for s in scores:
        if s > highest:
            highest = s

    lowest = scores[0]
    for s in scores:
        if s < lowest:
            lowest = s

    print("Average:", avg)
    print("Highest:", highest)
    print("Lowest:", lowest)
```

Expected Output:

- Split into functions: calculate_average, find_highest, find_lowest.
- Clean main process_scores() using helper functions.

Task Description #5 – Code Review on Error Handling

Task: Provide AI with this faulty code and ask it to improve error handling, naming, and readability.

Python Script

```
def div(a,b):
    return a/b
print(div(10,0))
```

Expected Output:

- Function with proper error handling using `try-except`.
- Better naming (`divide_numbers`).
- AI-generated docstring explaining error handling.

Task Description #6 – Complexity Reduction

Task: Use AI to simplify overly complex logic.

Sample Input Code:

```
def grade(score):
    if score >= 90:
        return "A"
    else:
        if score >= 80:
            return "B"
        else:
            if score >= 70:
                return "C"
            else:
                if score >= 60:
                    return "D"
                else:
                    return "F"
```

Expected Output:

- Cleaner logic using elif or dictionary mapping.

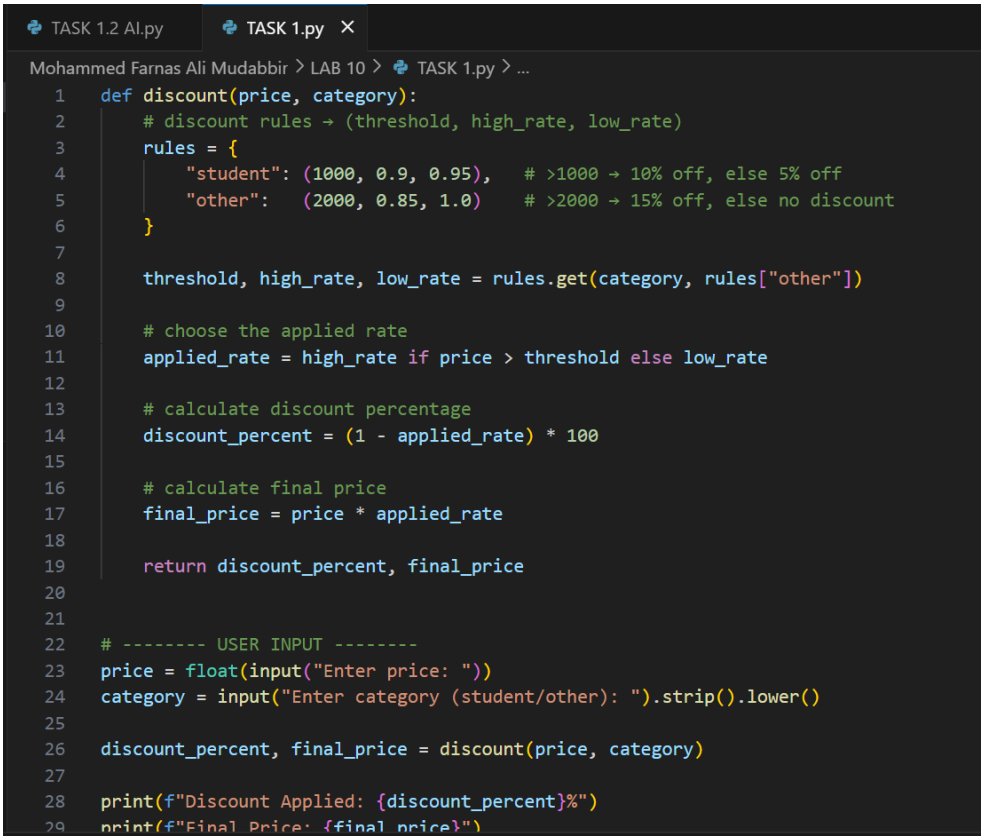
Task Description #1 – Refactor Nested Conditionals

Task: Provide AI with the following nested conditional code and ask it to simplify and refactor for readability.

Python script.

```
def discount(price, category):
    if category == "student":
        if price > 1000:
            return price * 0.9
        else:
            return price * 0.95
    else:
        if price > 2000:
            return price * 0.85
        else:
            return price
```

PROMPT: Refactor the following nested conditional code to make it cleaner and more readable. You may use simplified logic, dictionaries, or helper functions.



Expected Output:

- Refactored code using cleaner logic, possibly a dictionary or separate helper functions.

PRACTICAL OUTPUT:

Task Description #2 – Optimize Redundant Loops

Task: Give AI this messy loop and ask it to refactor and optimize.

Python script

```
def find_common(a, b):
    res = []
    for i in a:
        for j in b:
            if i == j:
                res.append(i)
    return res
```

PROMPT: Write a Python program where the user enters two lists. Then optimize the function find_common(a, b) that currently uses nested loops. Refactor it to a cleaner and faster version using sets (set(a) & set(b)). Show both the original and optimized outputs.

TASK 1.2 AI.py | TASK 1.py | TASK 2.py

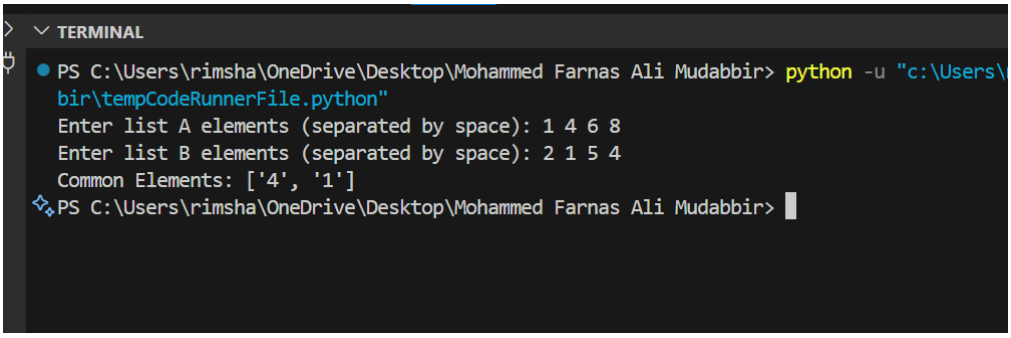
Mohammed Farnas Ali Mudabbir > LAB 10 > TASK 2.py > ...

```python
def find_common(a, b):
    return list(set(a) & set(b))


# -------- USER INPUT --------
a = input("Enter list A elements (separated by space): ").split()
b = input("Enter list B elements (separated by space): ").split()

common = find_common(a, b)

print("Common Elements:", common)
```

Expected Output:

Cleaner version using Python sets (set(a) & set(b))

PRACTICAL OUTPUT:



Task Description #3 – Improve Class Design

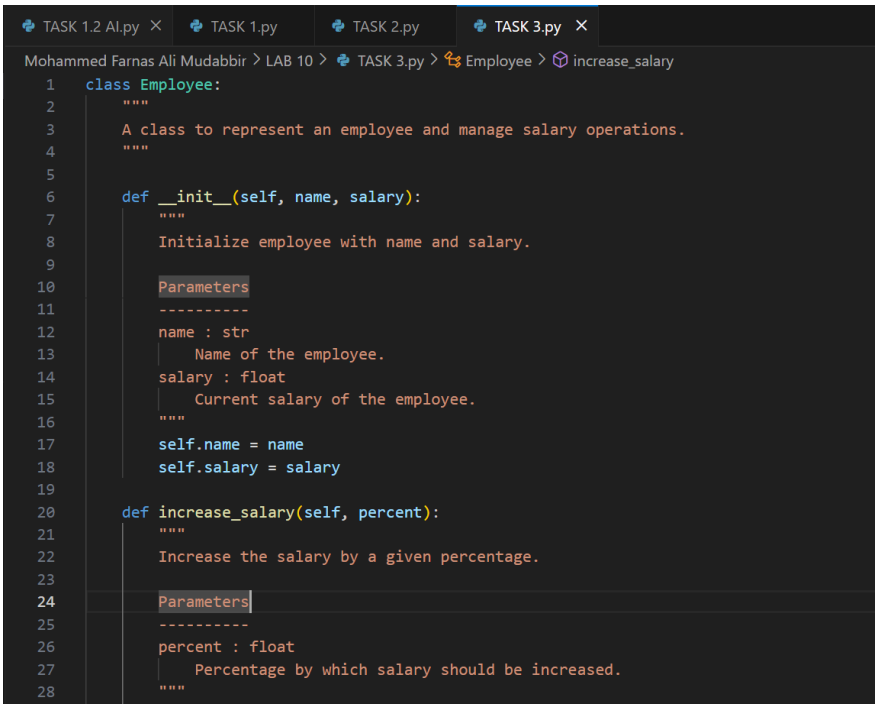
Task: Provide this class with poor readability and ask AI to improve:

- Naming conventions
- Encapsulation
- Readability & maintainability

Python Script

```
class emp:
    def __init__(self,n,s):
        self.n=n
        self.s=s
    def inc(self,p):
        self.s=self.s+(self.s*p/100)
    def pr(self):
        print("emp:",self.n,"salary:",self.s)
```

PROMPT: Write a Python program where the user enters an employee name and salary. Use this poorly written class and improve it:



TASK 1.2 AI.py | TASK 1.py | TASK 2.py | TASK 3.py

Mohammed Farnas Ali Mudabbir > LAB 10 > TASK 3.py > Employee > increase_salary

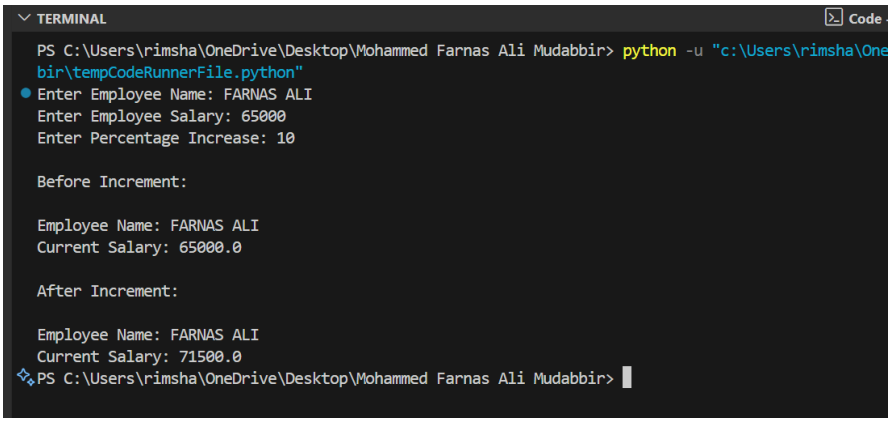
```
class Employee:
    def increase_salary(self, percent):
        """
        self.salary += self.salary * (percent / 100)

    def display_info(self):
        """
        Display employee details in formatted output.
        """
        print(f"\nEmployee Name: {self.name}")
        print(f"Current Salary: {self.salary}")


# ---------- USER INPUT ----------
name = input("Enter Employee Name: ")
salary = float(input("Enter Employee Salary: "))
percent = float(input("Enter Percentage Increase: "))

# Create object
emp = Employee(name, salary)

print("\nBefore Increment:")
emp.display_info()

# Apply salary increment
emp.increase_salary(percent)
```

```
46
47  print("\nBefore Increment:")
48  emp.display_info()
49
50  # Apply salary increment
51  emp.increase_salary(percent)
52
53  print("\nAfter Increment:")
54  emp.display_info()
55
```

Expected Output:

Employee class with meaningful methods (increase_salary, display_info), formatted output, and added docstrings

PRACTICAL OUTPUT:



Task Description #4 – Modularize Long Function

Task: Give AI this long unstructured function and let it modularize into smaller helper functions.

Python Script

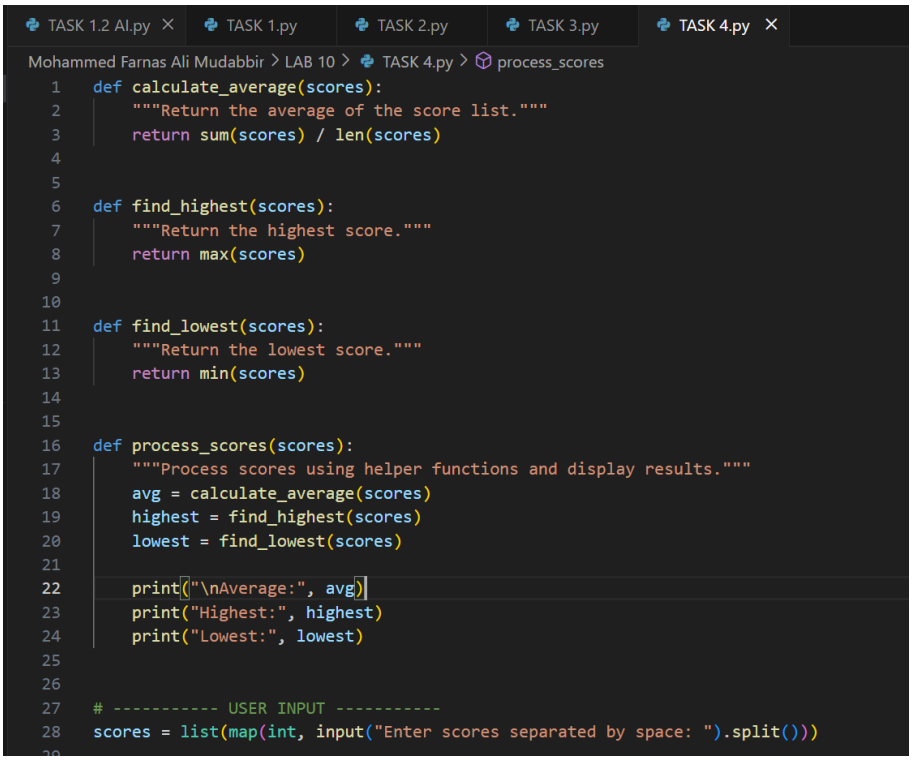
```
def process_scores(scores):
    total = 0
    for s in scores:
        total += s
    avg = total / len(scores)

    highest = scores[0]
    for s in scores:
        if s > highest:
            highest = s

    lowest = scores[0]
    for s in scores:
        if s < lowest:
            lowest = s
```

print("Average:", avg)

print("Highest:", highest)

print("Lowest:", lowest)

PROMPT: Take user input for a list of scores and refactor the process_scores() function by splitting it into helper functions: calculate_average(), find_highest(), and find_lowest(). Then rewrite the main function using these helpers.

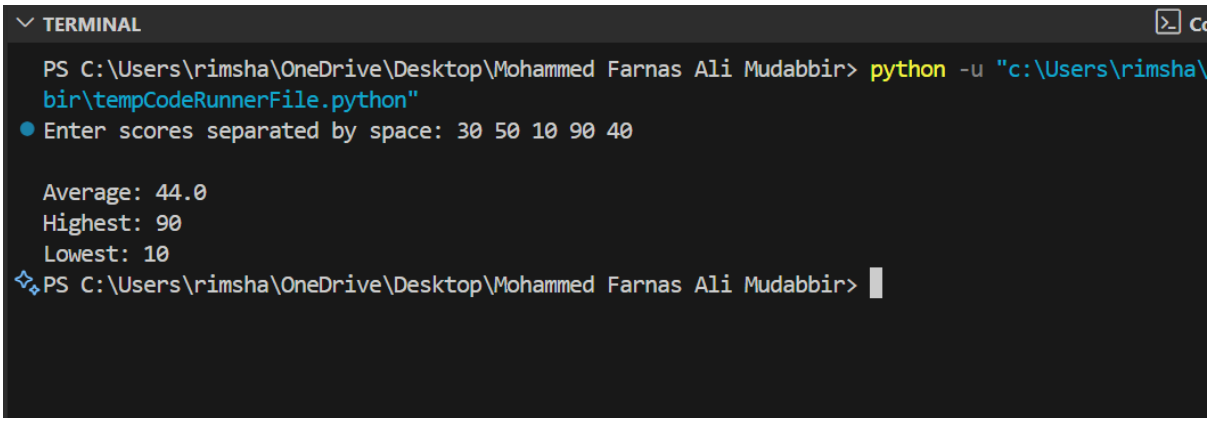


Expected Output:

- Split into functions: calculate_average, find_highest, find_lowest.

Clean main process_scores() using helper functions.

PRACTICAL OUTPUT:



Task Description #5 – Code Review on Error Handling

Task: Provide AI with this faulty code and ask it to improve error handling, naming, and readability.

Python Script

```
def div(a,b):
    return a/b
print(div(10,0))
```

PROMPT: Improve this code by adding error handling, better naming, and a docstring

TASK 1.2 AI.py | TASK 1.py | TASK 2.py | TASK 3.py | TASK 4.py | def divide_numbers(a, b): U

```python
def divide_numbers(a, b):
    """
    Divide two numbers safely with error handling.

    Parameters
    ----------
    a : float or int
        The numerator.
    b : float or int
        The denominator.

    Returns
    -------
    float or str
        The result of division, or an error message if division fails.

    Notes
    -----
    This function uses a try-except block to catch ZeroDivisionError
    and return a user-friendly message instead of crashing the program.
    """
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."


print(divide_numbers(10, 0))
```

Expected Output:

- Function with proper error handling using `try-except`.
- Better naming (`divide_numbers`).
- AI-generated docstring explaining error handling.

PRACTICAL OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

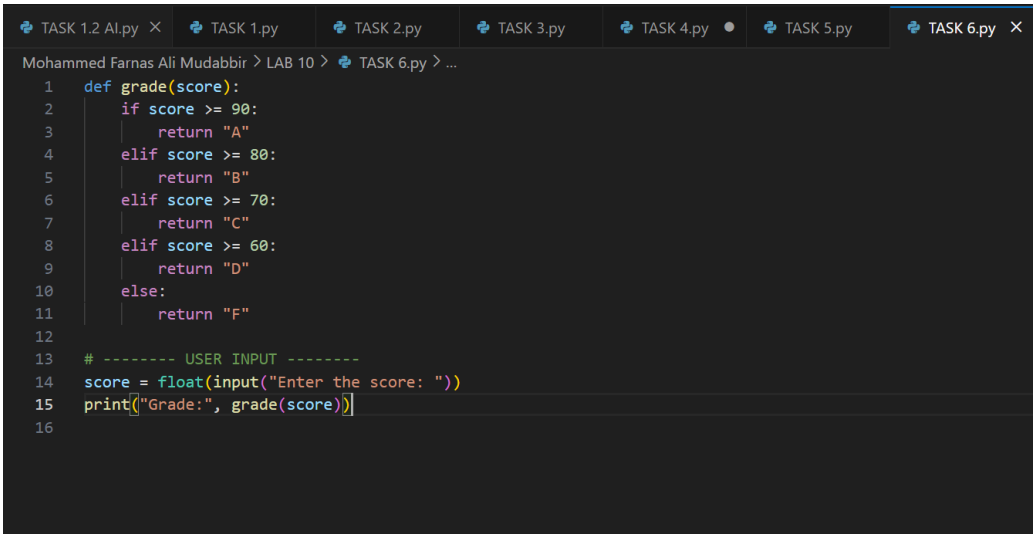
```
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\
bir\tempCodeRunnerFile.python"
Error: Division by zero is not allowed.
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir>
```

Task Description #6 – Complexity Reduction

Task: Use AI to simplify overly complex logic.
Sample Input Code:

```
def grade(score):
    if score >= 90:
        return "A"
    else:
        if score >= 80:
            return "B"
        else:
            if score >= 70:
                return "C"
            else:
                if score >= 60:
                    return "D"
                else:
                    return "F"
```

PROMPT: Write a Python program that takes **user input for score** and simplifies the nested grade() function. Replace the deep nested if-else structure with cleaner logic using **elif** or a **dictionary**.



Expected Output:

- Cleaner logic using elif or dictionary mapping.
-

PRACTICAL OUTPUT: