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:
		Week5 - Monday

```
        return price * 0.95
    else:
        if price > 2000:
            return price * 0.85
        else:
            return price
```

Expected Output:

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

CODE:

```
Users > Anjali > 10.1.py
def student_discount(price):
    return price * 0.9 if price > 1000 else price * 0.95

def regular_discount(price):
    return price * 0.85 if price > 2000 else price

def discount(price, category):
    discount_fn = {
        "student": student_discount,
        "regular": regular_discount
    }
    return discount_fn.get(category, regular_discount)(price)

# Take input from user
try:
    price = float(input("Enter the price: "))
    category = input("Enter the category (student/regular): ").strip().lower()
    final_price = discount(price, category)
    print(f"Discounted price: {final_price:.2f}")
except ValueError:
    print("Invalid input. Please enter a numeric value for price.")
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter the price: 1200
Enter the category (student/regular): student
Discounted price: 1080.00
```

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))

CODE:

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

# Taking input from the user
list1 = list(map(int, input("Enter elements of List 1 separated by space: ").split()))
list2 = list(map(int, input("Enter elements of List 2 separated by space: ").split()))

# Calling the function
result = find_common(list1, list2)

# Output
print("List 1:", list1)
print("List 2:", list2)
print("Common elements:", result)
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python>
& "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter elements of List 1 separated by space: 1 2 3 4 5 6
Enter elements of List 2 separated by space: 4 5 6 7 8
List 1: [1, 2, 3, 4, 5, 6]
List 2: [4, 5, 6, 7, 8]
Common elements: [4, 5, 6]
```

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.

CODE:

```
class Employee:
    """
    Represents an employee with a name and salary.
    Provides functionality to increase salary and display employee information.
    """
    def __init__(self, name, salary):
        self._name = name
        self._salary = salary

    def increase_salary(self, percent):
        """
        Increases the employee's salary by a given percentage.

        Args:
            percent (float): The percentage increase to apply.
        """
        self._salary += self._salary * percent / 100

    def display_info(self):
        """
        Displays the employee's name and current salary.
        """
        print(f"Employee: {self._name}, Salary: ₹{self._salary:.2f}")


# Take input from user
try:
    name = input("Enter employee name: ")
    salary = float(input("Enter current salary: "))
    percent = float(input("Enter percentage increase: "))

    emp = Employee(name, salary)
    emp.increase_salary(percent)
    emp.display_info()

except ValueError:
    print("Invalid input. Please enter numeric values for salary and percentage.")
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python>
& "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter employee name: Mohammed Nizamuddin
Enter current salary: 45000
Enter percentage increase: 7.5
Employee: Mohammed Nizamuddin, Salary: ₹48375.00
```

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.

CODE:

```python
def calculate_average(scores):
    return sum(scores) / len(scores)

def find_highest(scores):
    return max(scores)

def find_lowest(scores):
    return min(scores)

def process_scores(scores):
    average = calculate_average(scores)
    highest = find_highest(scores)
    lowest = find_lowest(scores)

    print(f"Average: {average:.2f}")
    print(f"Highest: {highest}")
    print(f"Lowest: {lowest}")

# Take input from user
try:
    input_scores = input("Enter scores separated by commas: ")
    scores = [float(s.strip()) for s in input_scores.split(",")]
    process_scores(scores)
except ValueError:
    print("Invalid input. Please enter numeric scores separated by commas.")
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python>
& "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter scores separated by commas: 10,12,18,27,31
Average: 19.60
Highest: 31.0
Lowest: 10.0
```

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.

CODE:

```
def divide_numbers(a, b):
    """
    Divides two numbers and handles division by zero.

    Args:
        a (float): Numerator.
        b (float): Denominator.

    Returns:
        float: Result of division if valid.
        str: Error message if division by zero occurs.
    """
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."

# Take input from user
try:
    a = float(input("Enter the numerator: "))
    b = float(input("Enter the denominator: "))
    result = divide_numbers(a, b)
    print("Result:", result)
except ValueError:
    print("Invalid input. Please enter numeric values.")
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python>
& "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter the numerator: 15
Enter the denominator: 3
Result: 5.0
```

Task Description #6 – Complexity Reduction

Task: Use AI to simplify overly complex logic.

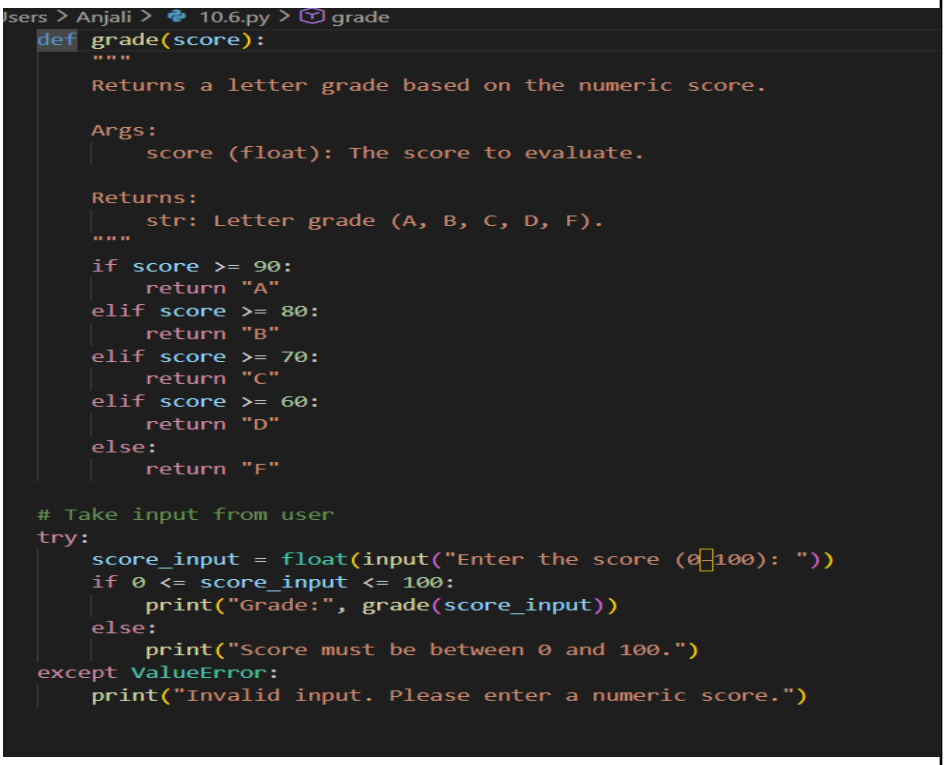
Sample Input Code:

```
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.

CODE:



```
sers > Anjali > 10.6.py > grade
def grade(score):
    """
    Returns a letter grade based on the numeric score.

    Args:
        score (float): The score to evaluate.

    Returns:
        str: Letter grade (A, B, C, D, F).
    """
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"

# Take input from user
try:
    score_input = float(input("Enter the score (0-100): "))
    if 0 <= score_input <= 100:
        print("Grade:", grade(score_input))
    else:
        print("Score must be between 0 and 100.")
except ValueError:
    print("Invalid input. Please enter a numeric score.")
```

OUTPUT:

```
PS C:\Users\moham\Desktop\Python>
& "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter the score (0–100): 84
Grade: B
```