

| 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   | <p><b>Lab 10 – Code Review and Quality: Using AI to Improve Code Quality and Readability</b></p> <p><b>Lab Objectives</b></p> <ul style="list-style-type: none"> <li>• Use AI for automated code review and quality enhancement.</li> <li>• Identify and fix syntax, logical, performance, and security issues in Python code.</li> <li>• Improve readability and maintainability through structured refactoring and comments.</li> <li>• Apply prompt engineering for targeted improvements.</li> <li>• Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices</li> </ul> <hr/> <p><b>Task Description #1 – Refactor Nested Conditionals</b></p> <p>Task: Provide AI with the following nested conditional code and ask it to simplify and refactor for readability.</p> <p><b>Python script.</b></p> <pre>def discount(price, category):     if category == "student":         if price &gt; 1000:             return price * 0.9         else:             return price * 0.95     else:         if price &gt; 2000:             return price * 0.85</pre> |  | 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.
-

|  |  |  |
|--|--|--|
|  | <p><b>Task Description #4 – Modularize Long Function</b></p> <p>Task: Give AI this long unstructured function and let it modularize into smaller helper functions.</p> <p><b>Python Script</b></p> <pre>def process_scores(scores):<br/>    total = 0<br/>    for s in scores:<br/>        total += s<br/>    avg = total / len(scores)<br/><br/>    highest = scores[0]<br/>    for s in scores:<br/>        if s &gt; highest:<br/>            highest = s<br/><br/>    lowest = scores[0]<br/>    for s in scores:<br/>        if s &lt; lowest:<br/>            lowest = s<br/><br/>    print("Average:", avg)<br/>    print("Highest:", highest)<br/>    print("Lowest:", lowest)</pre> <p><b>Expected Output:</b></p> <ul style="list-style-type: none"><li>• Split into functions: calculate_average, find_highest, find_lowest.</li><li>• Clean main process_scores() using helper functions.</li></ul> <hr/> <p><b>Task Description #5 – Code Review on Error Handling</b></p> <p>Task: Provide AI with this faulty code and ask it to improve error handling, naming, and readability.</p> <p><b>Python Script</b></p> |  |
|--|--|--|

|  |  |  |
|--|--|--|
|  | <pre>def div(a,b):<br/>    return a/b<br/>print(div(10,0))</pre> <p><b>Expected Output:</b></p> <ul style="list-style-type: none"><li>• Function with proper error handling using try-except.</li><li>• Better naming (divide_numbers).</li><li>• AI-generated docstring explaining error handling.</li></ul> <hr/> <p><b>Task Description #6 – Complexity Reduction</b><br/>Task: Use AI to simplify overly complex logic.<br/>Sample Input Code:</p> <pre>def grade(score):<br/>    if score &gt;= 90:<br/>        return "A"<br/>    else:<br/>        if score &gt;= 80:<br/>            return "B"<br/>        else:<br/>            if score &gt;= 70:<br/>                return "C"<br/>            else:<br/>                if score &gt;= 60:<br/>                    return "D"<br/>                else:<br/>                    return "F"</pre> <p>Expected Output:</p> <ul style="list-style-type: none"><li>• Cleaner logic using elif or dictionary mapping.</li></ul> |  |
|--|--|--|

## 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
```

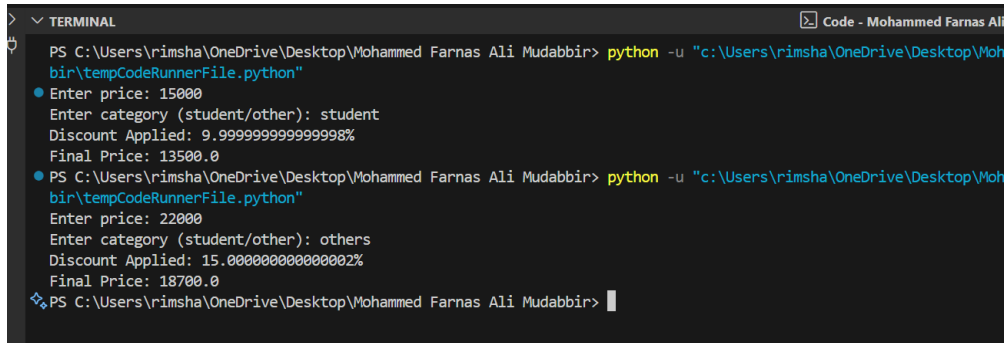
**PROMPT:** Refactor the following nested conditional code to make it cleaner and more readable. You may use simplified logic, dictionaries, or helper functions.

```
TASK 1.2 AI.py  TASK 1.py X
Mohammed Farnas Ali Mudabbir > LAB 10 > TASK 1.py > ...
1  def discount(price, category):
2      # discount rules → (threshold, high_rate, low_rate)
3      rules = {
4          "student": (1000, 0.9, 0.95), # >1000 → 10% off, else 5% off
5          "other": (2000, 0.85, 1.0)   # >2000 → 15% off, else no discount
6      }
7
8      threshold, high_rate, low_rate = rules.get(category, rules["other"])
9
10     # choose the applied rate
11     applied_rate = high_rate if price > threshold else low_rate
12
13     # calculate discount percentage
14     discount_percent = (1 - applied_rate) * 100
15
16     # calculate final price
17     final_price = price * applied_rate
18
19     return discount_percent, final_price
20
21
22     # ----- USER INPUT -----
23     price = float(input("Enter price: "))
24     category = input("Enter category (student/other): ").strip().lower()
25
26     discount_percent, final_price = discount(price, category)
27
28     print(f"Discount Applied: {discount_percent}%")
29     print(f"Final Price: {final_price}")
```

### Expected Output:

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

### PRACTICAL OUTPUT:



```
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\tempCodeRunnerFile.python"
Enter price: 15000
Enter category (student/other): student
Discount Applied: 9.999999999999998%
Final Price: 13500.0
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\tempCodeRunnerFile.python"
Enter price: 22000
Enter category (student/other): others
Discount Applied: 15.000000000000002%
Final Price: 18700.0
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir>
```

## 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 Al.py TASK 1.py TASK 2.py X
Mohammed Farnas Ali Mudabbir > LAB 10 > TASK 2.py > ...
1  def find_common(a, b):
2      return list(set(a) & set(b))
3
4
5  # ----- USER INPUT -----
6  a = input("Enter list A elements (separated by space): ").split()
7  b = input("Enter list B elements (separated by space): ").split()
8
9  common = find_common(a, b)
10
11 print("Common Elements:", common)
12
```

### Expected Output:

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

### PRACTICAL OUTPUT:

```
> ▼ TERMINAL
● PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive\Desktop\tempCodeRunnerFile.py"
Enter list A elements (separated by space): 1 4 6 8
Enter list B elements (separated by space): 2 1 5 4
Common Elements: ['4', '1']
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir>
```

## Task Description #3 – Improve Class Design

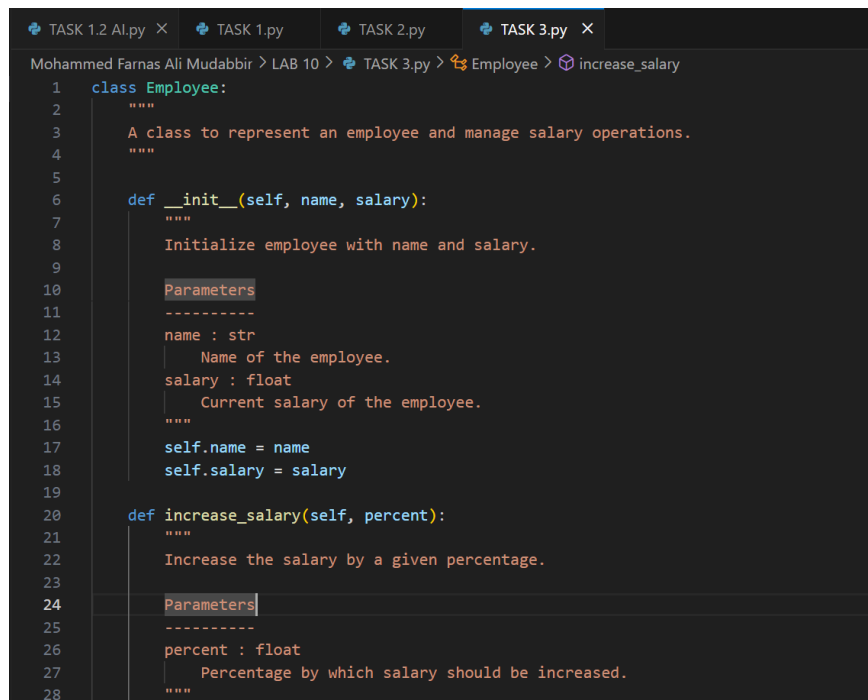
**Task:** Provide this class with poor readability and ask AI to improve:

- 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 x TASK 1.py TASK 2.py TASK 3.py x
Mohammed Farnas Ali Mudabbir > LAB 10 > TASK 3.py > Employee > increase_salary
1 class Employee:
2     """
3     A class to represent an employee and manage salary operations.
4     """
5
6     def __init__(self, name, salary):
7         """
8         Initialize employee with name and salary.
9
10        Parameters
11        -----
12        name : str
13            Name of the employee.
14        salary : float
15            Current salary of the employee.
16        """
17        self.name = name
18        self.salary = salary
19
20    def increase_salary(self, percent):
21        """
22        Increase the salary by a given percentage.
23
24        Parameters
25        -----
26        percent : float
27            Percentage by which salary should be increased.
28        """
```



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

Mohammed Farnas Ali Mudabbir > LAB 10 > TASK 3.py > Employee > increase\_salary

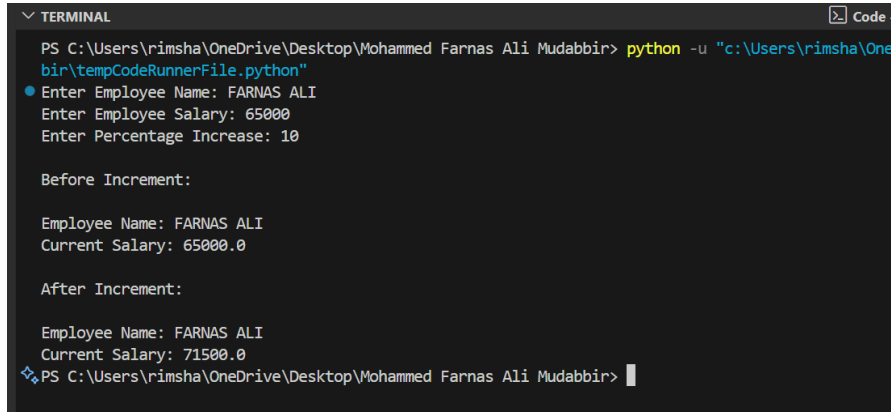
```
1 class Employee:
20     def increase_salary(self, percent):
28         """
29         self.salary += self.salary * (percent / 100)
30
31     def display_info(self):
32         """
33         Display employee details in formatted output.
34         """
35         print(f"\nEmployee Name: {self.name}")
36         print(f"Current Salary: {self.salary}")
37
38
39 # ----- USER INPUT -----
40 name = input("Enter Employee Name: ")
41 salary = float(input("Enter Employee Salary: "))
42 percent = float(input("Enter Percentage Increase: "))
43
44 # Create object
45 emp = Employee(name, salary)
46
47 print("\nBefore Increment:")
48 emp.display_info()
49
50 # Apply salary increment
51 emp.increase_salary(percent)
52
```

```
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:



```
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive\Desktop\tempCodeRunnerFile.py"
Enter Employee Name: FARNAS ALI
Enter Employee Salary: 65000
Enter Percentage Increase: 10

Before Increment:

Employee Name: FARNAS ALI
Current Salary: 65000.0

After Increment:

Employee Name: FARNAS ALI
Current Salary: 71500.0
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir>
```

## 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.

```
TASK 1.2 AI.py X TASK 1.py TASK 2.py TASK 3.py TASK 4.py X
Mohammed Farnas Ali Mudabbir > LAB 10 > TASK 4.py > process_scores
1  def calculate_average(scores):
2      """Return the average of the score list."""
3      return sum(scores) / len(scores)
4
5
6  def find_highest(scores):
7      """Return the highest score."""
8      return max(scores)
9
10
11 def find_lowest(scores):
12     """Return the lowest score."""
13     return min(scores)
14
15
16 def process_scores(scores):
17     """Process scores using helper functions and display results."""
18     avg = calculate_average(scores)
19     highest = find_highest(scores)
20     lowest = find_lowest(scores)
21
22     print("\nAverage:", avg)
23     print("Highest:", highest)
24     print("Lowest:", lowest)
25
26
27 # ----- USER INPUT -----
28 scores = list(map(int, input("Enter scores separated by space: ").split()))
29
30
31
21
22     print("\nAverage:", avg)
23     print("Highest:", highest)
24     print("Lowest:", lowest)
25
26
27 # ----- USER INPUT -----
28 scores = list(map(int, input("Enter scores separated by space: ").split()))
29
30 process_scores(scores)
31
```

### Expected Output:

- Split into functions: calculate\_average, find\_highest, find\_lowest.

Clean main process\_scores() using helper functions.

### PRACTICAL OUTPUT:

```
▼ TERMINAL
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\
bir\tempCodeRunnerFile.python"
● Enter scores separated by space: 30 50 10 90 40

Average: 44.0
Highest: 90
Lowest: 10
❖ PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> |
```

### 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
1 def divide_numbers(a, b):
2     """
3     Divide two numbers safely with error handling.
4
5     Parameters
6     -----
7     a : float or int
8         The numerator.
9     b : float or int
10        The denominator.
11
12    Returns
13    -----
14    float or str
15        The result of division, or an error message if division fails.
16
17    Notes
18    ----
19    This function uses a try-except block to catch ZeroDivisionError
20    and return a user-friendly message instead of crashing the program.
21    """
22    try:
23        return a / b
24    except ZeroDivisionError:
25        return "Error: Division by zero is not allowed."
26
27
28 print(divide_numbers(10, 0))
29 |
```

### 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\rimsha\OneDrive\Desktop\tempCodeRunnerFile.py"
Error: Division by zero is not allowed.
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir>
```

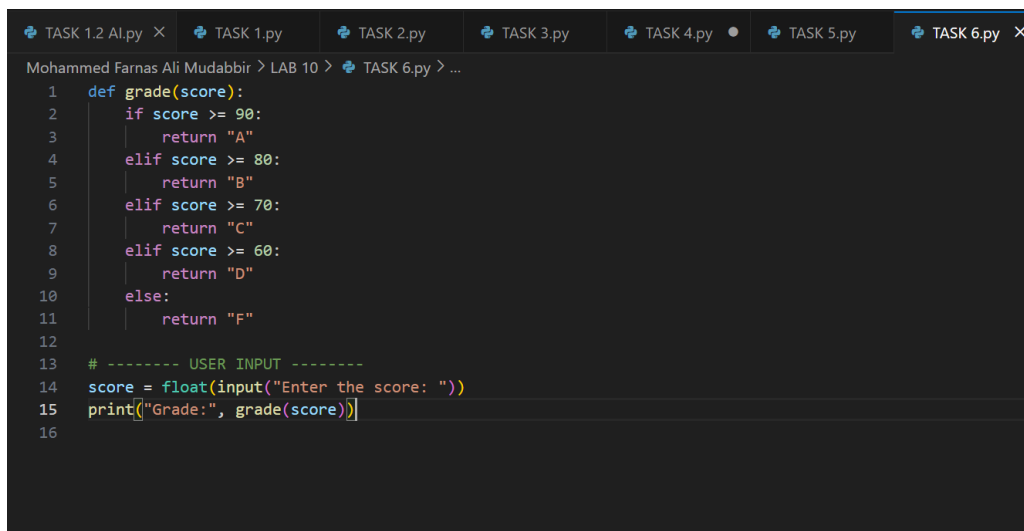
## Task Description #6 – Complexity Reduction

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**.



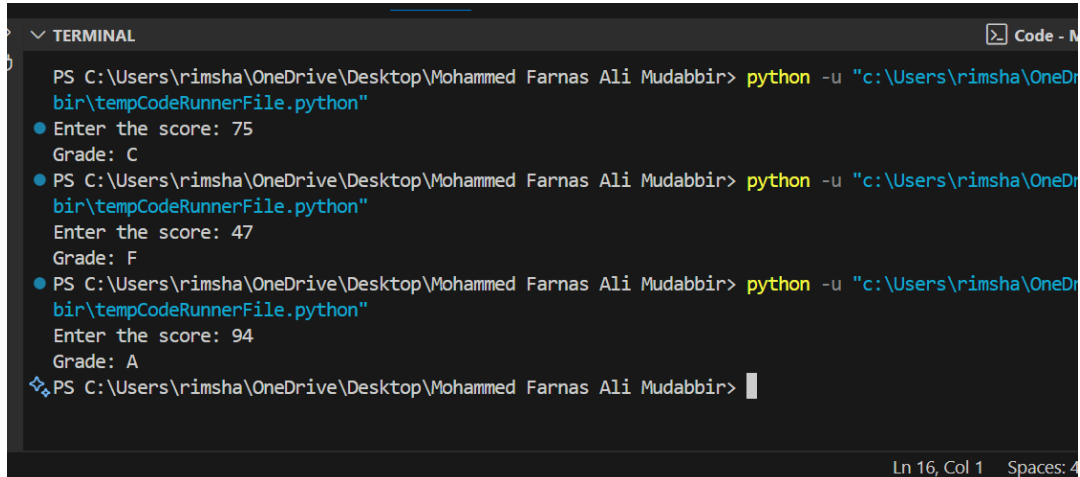
The screenshot shows a code editor with a dark background. At the top, there are several tabs labeled 'TASK 1.2 Al.py', 'TASK 1.py', 'TASK 2.py', 'TASK 3.py', 'TASK 4.py', 'TASK 5.py', and 'TASK 6.py'. The 'TASK 6.py' tab is active. Below the tabs, the editor displays the following Python code:

```
1  def grade(score):
2      if score >= 90:
3          return "A"
4      elif score >= 80:
5          return "B"
6      elif score >= 70:
7          return "C"
8      elif score >= 60:
9          return "D"
10     else:
11         return "F"
12
13 # ----- USER INPUT -----
14 score = float(input("Enter the score: "))
15 print("Grade:", grade(score))
16
```

### Expected Output:

- Cleaner logic using elif or dictionary mapping.
- 

### PRACTICAL OUTPUT:



```
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\tempCodeRunnerFile.python"
Enter the score: 75
Grade: C
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\tempCodeRunnerFile.python"
Enter the score: 47
Grade: F
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir> python -u "c:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\tempCodeRunnerFile.python"
Enter the score: 94
Grade: A
PS C:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir>
```

Ln 16, Col 1 Spaces: 4