

AI ASSISTED CODING

Assignment-3.2

P Sree Varshini

2303A52336

Batch-40

Task Description-1

- Progressive Prompting for Calculator Design: Ask the AI to design a simple calculator

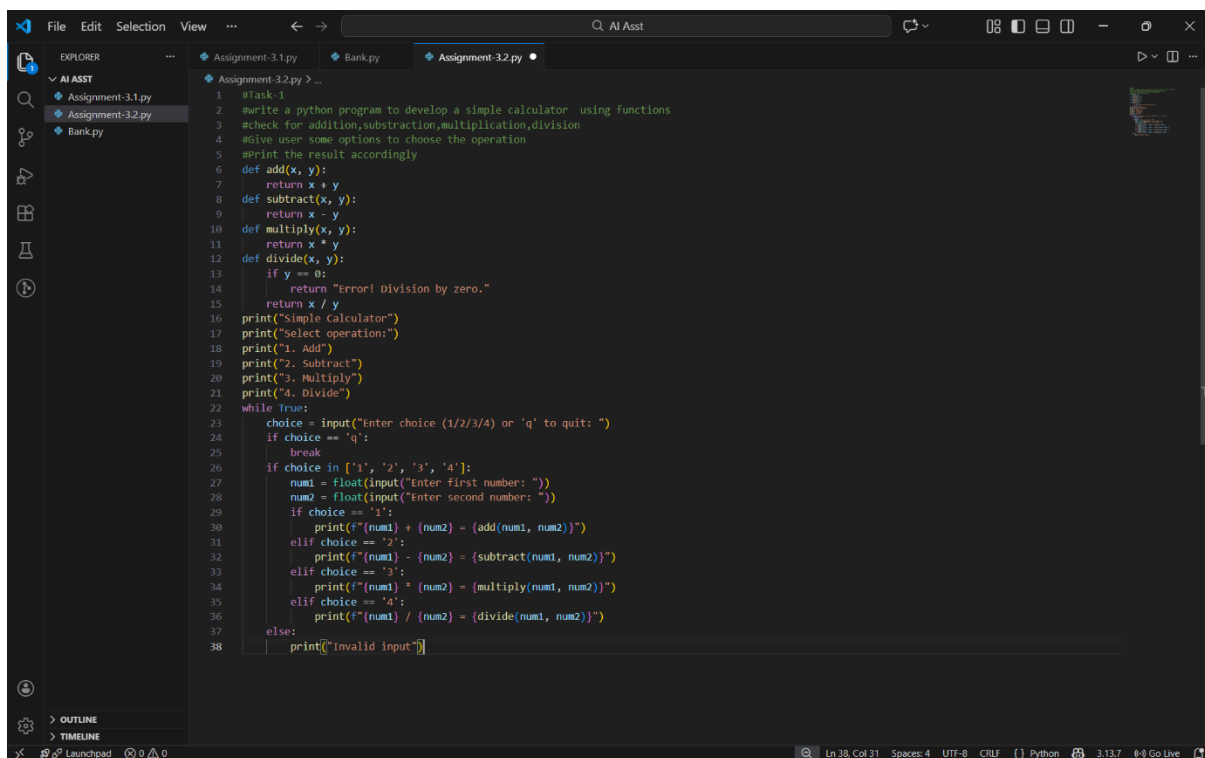
program by initially providing only the function name. Gradually enhance the prompt by

adding comments and usage examples.

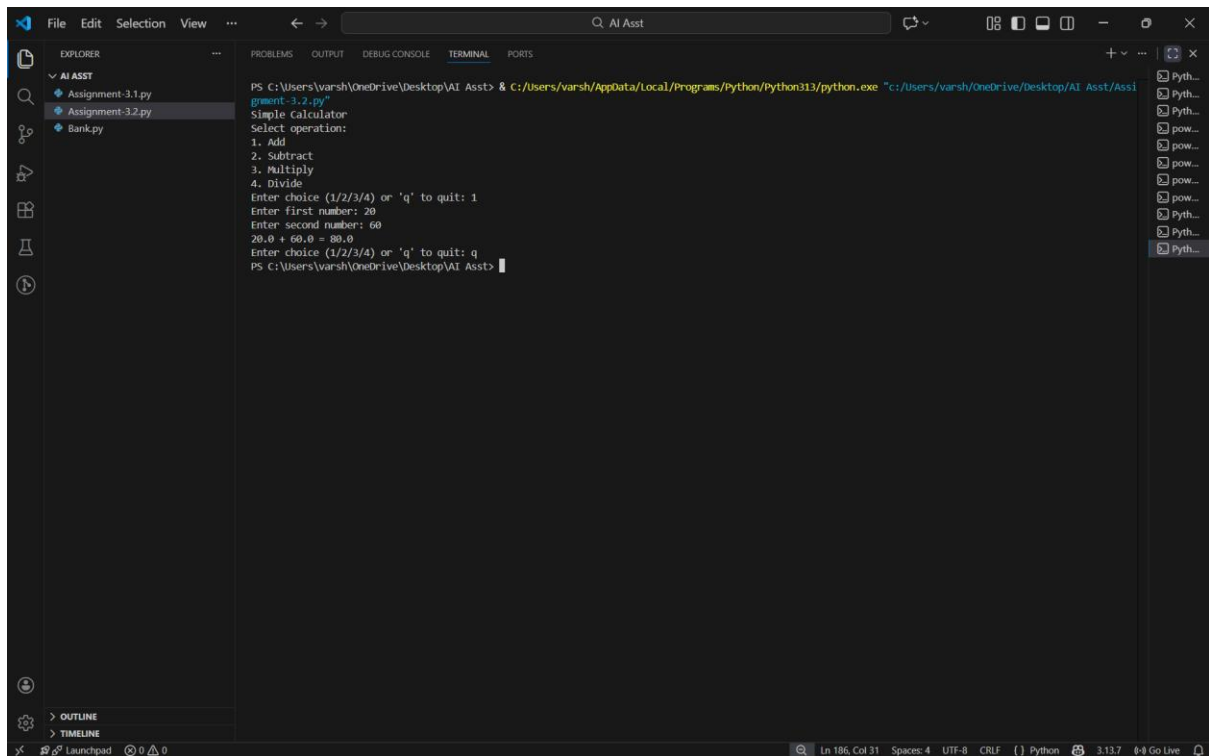
Expected Output-1

- Comparison showing improvement in AI-generated calculator logic and structure.

Code and Output:

A screenshot of a Visual Studio Code editor window. The Explorer sidebar on the left shows a project named 'AI ASST' with files 'Assignment-3.1.py', 'Assignment-3.2.py', and 'Bank.py'. The main editor area displays the code for 'Assignment-3.2.py'. The code is a Python script for a simple calculator. It includes comments in Hindi explaining the purpose of the program and the operations to be implemented. The code defines four functions: add, subtract, multiply, and divide. It then uses a while loop to prompt the user to choose an operation (1 for Add, 2 for Subtract, 3 for Multiply, 4 for Divide) or 'q' to quit. Based on the user's choice, it prompts for two numbers and performs the corresponding operation, displaying the result. The status bar at the bottom indicates the file is at line 38, column 31, with 4 spaces, UTF-8 encoding, CRLF line endings, and is a Python file using Python 3.13.7 interpreter.

```
1 #task-1
2 #write a python program to develop a simple calculator using functions
3 #check for addition, subtraction, multiplication, division
4 #give user some options to choose the operation
5 #print the result accordingly
6 def add(x, y):
7     return x + y
8 def subtract(x, y):
9     return x - y
10 def multiply(x, y):
11     return x * y
12 def divide(x, y):
13     if y == 0:
14         return "Error! Division by zero."
15     return x / y
16 print("Simple calculator")
17 print("Select operation:")
18 print("1. Add")
19 print("2. Subtract")
20 print("3. Multiply")
21 print("4. Divide")
22 while True:
23     choice = input("Enter choice (1/2/3/4) or 'q' to quit: ")
24     if choice == 'q':
25         break
26     if choice in ['1', '2', '3', '4']:
27         num1 = float(input("Enter first number: "))
28         num2 = float(input("Enter second number: "))
29         if choice == '1':
30             print(f"{num1} + {num2} = {add(num1, num2)}")
31         elif choice == '2':
32             print(f"{num1} - {num2} = {subtract(num1, num2)}")
33         elif choice == '3':
34             print(f"{num1} * {num2} = {multiply(num1, num2)}")
35         elif choice == '4':
36             print(f"{num1} / {num2} = {divide(num1, num2)}")
37     else:
38         print("Invalid input")
```



```
PS C:\Users\varsh\OneDrive\Desktop\AI Asst> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/varsh/OneDrive/Desktop/AI Asst/Assignment-3.2.py"
Simple Calculator
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1/2/3/4) or 'q' to quit: 1
Enter first number: 20
Enter second number: 60
20.0 + 60.0 = 80.0
Enter choice (1/2/3/4) or 'q' to quit: q
PS C:\Users\varsh\OneDrive\Desktop\AI Asst>
```

Analysis:

Analysis: Progressive Prompting for Calculator Design

Progressive prompting means giving instructions step by step to the AI.

First, only the function name is given, so the output is basic.

Next, comments are added to explain what the calculator should do.

Then, usage examples are included to show how the program works.

Each step improves clarity and accuracy of the AI response.

This method helps in getting a complete and correct calculator program.

Task Description-2

- Refining Prompts for Sorting Logic: Start with a vague prompt for sorting student marks,

then refine it to clearly specify sorting order and constraints.

Expected Output-2

- AI-generated sorting function evolves from ambiguous logic to an accurate and efficient implementation.

Code and Output:

The screenshot shows a VS Code editor with a file named 'Assignment-3.2.py'. The code is a Python program designed to sort student marks. It includes comments in Hindi explaining the task and constraints. The program takes user input for marks, sorts them, and prints the sorted list. It also includes a validation check for marks between 0 and 100.

```

40 #Task-2
41 #Write a python program to sort the students marks print after soring
42 #The code should have dynamic input from the user
43 #Constraints:Ensure all marks are between 0 and 100
44 #Print the output after sorting
45 def sort_marks(marks_list):
46     return sorted(marks_list)
47 user_input = input("Enter student marks separated by spaces: ")
48 marks_list = list(map(int, user_input.split()))
49 sorted_marks = sort_marks(marks_list)
50 print("Sorted student marks:", sorted_marks)
51 for mark in marks_list:
52     if mark < 0 or mark > 100:
53         print(f"Invalid mark found: {mark}. Marks should be between 0 and 100.")
54         break
55     else:
56         print("All marks are valid.")
57

```

The terminal output shows the execution of the program. It prompts the user to enter marks, and the user enters '60 40 20 90 80'. The program outputs the sorted marks: '[20, 40, 60, 80, 90]'. It also confirms that all marks are valid.

```

23.0 + 33.0 = 56.0
Enter choice (1/2/3/4) or 'q' to quit: q
PS C:\Users\varsh\OneDrive\Desktop\AI Asst> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe "C:/Users/varsh/OneDrive/Desktop/AI Asst/Assignment-3.2.py"
Enter student marks separated by spaces: 60 40 20 90 80
Sorted student marks: [20, 40, 60, 80, 90]
All marks are valid.
PS C:\Users\varsh\OneDrive\Desktop\AI Asst>

```

Analysis:

Refining Prompts for Sorting Logic

Initially, a vague prompt is given to sort student marks.

Due to less clarity, the AI may give an incomplete or unclear solution.

The prompt is then refined to specify sorting order (ascending or descending).

Additional constraints (such as valid marks or number of students) are added.

Clear instructions help the AI produce accurate and expected output.

Refining prompts reduces confusion and errors in the solution.

Task Description-3

- Few-Shot Prompting for Prime Number Validation: Provide multiple input-output

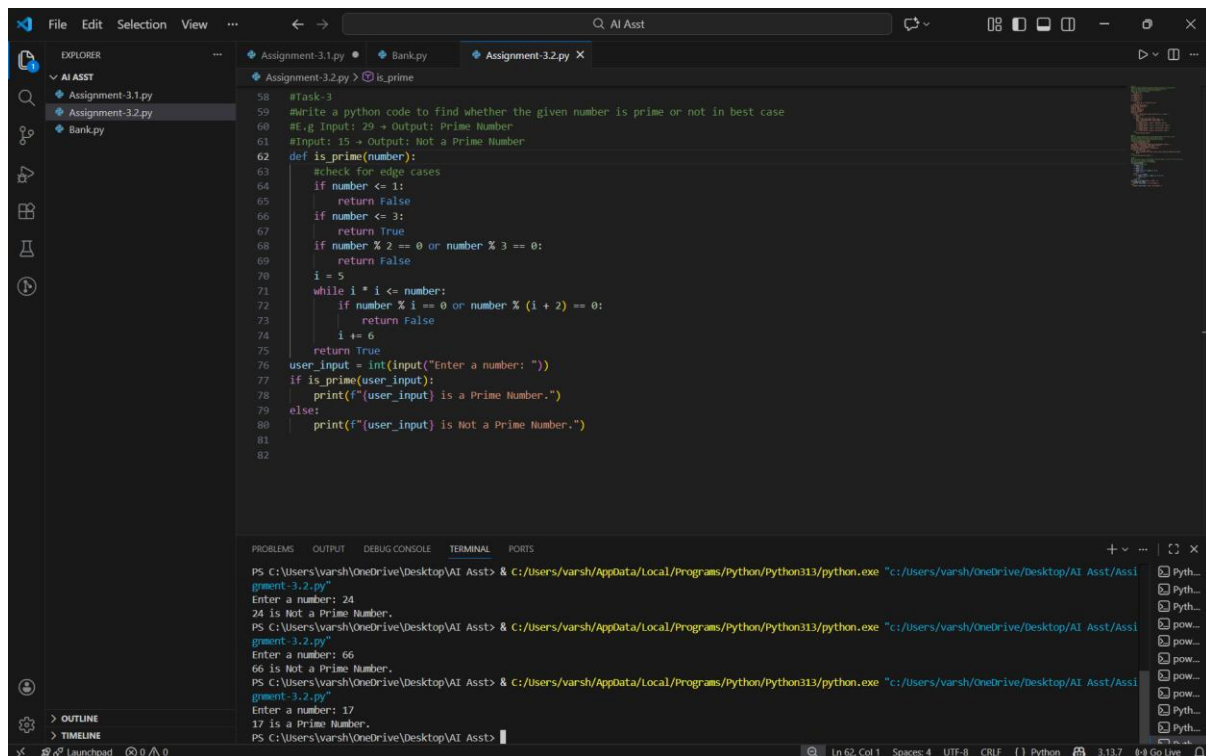
examples for a function that checks whether a number is prime. Observe how few-shot

prompting improves correctness.

Expected Output-3

- Improved prime-checking function with better edge-case handling.

Code and Output:



The screenshot shows a VS Code editor with a Python file named 'Assignment-3.2.py'. The code defines a function 'is_prime' that checks for prime numbers, handling edge cases like numbers less than 2 and divisibility by 2 and 3. It then prompts the user for input and prints the result. The terminal output shows three test cases: 24 (not prime), 66 (not prime), and 17 (prime).

```
58 #Task-3
59 #Write a python code to find whether the given number is prime or not in best case
60 #E.g Input: 29 → Output: Prime Number
61 #Input: 15 → Output: Not a Prime Number
62 def is_prime(number):
63     #check for edge cases
64     if number <= 1:
65         return False
66     if number <= 3:
67         return True
68     if number % 2 == 0 or number % 3 == 0:
69         return False
70     i = 5
71     while i * i <= number:
72         if number % i == 0 or number % (i + 2) == 0:
73             return False
74         i += 6
75     return True
76 user_input = int(input("Enter a number: "))
77 if is_prime(user_input):
78     print(f"{user_input} is a Prime Number.")
79 else:
80     print(f"{user_input} is Not a Prime Number.")
81
82
```

Terminal Output:

```
PS C:\Users\varsh\OneDrive\Desktop\AI Asst> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe "C:/Users/varsh/OneDrive/Desktop/AI Asst/Assignment-3.2.py"
Enter a number: 24
24 is Not a Prime Number.
PS C:\Users\varsh\OneDrive\Desktop\AI Asst> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe "C:/Users/varsh/OneDrive/Desktop/AI Asst/Assignment-3.2.py"
Enter a number: 66
66 is Not a Prime Number.
PS C:\Users\varsh\OneDrive\Desktop\AI Asst> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe "C:/Users/varsh/OneDrive/Desktop/AI Asst/Assignment-3.2.py"
Enter a number: 17
17 is a Prime Number.
PS C:\Users\varsh\OneDrive\Desktop\AI Asst>
```

Analysis:

Few-Shot Prompting for Prime Number Validation

Few-shot prompting means giving multiple input-output examples to the AI.

The examples show how prime and non-prime numbers behave.

This helps the AI understand the exact logic of prime number checking.

The AI produces more correct and reliable results.

Errors are reduced compared to a prompt with no examples.

Few-shot prompting improves accuracy and consistency of the function.

Task Description-4

- Prompt-Guided UI Design for Student Grading System: Create a user interface for a

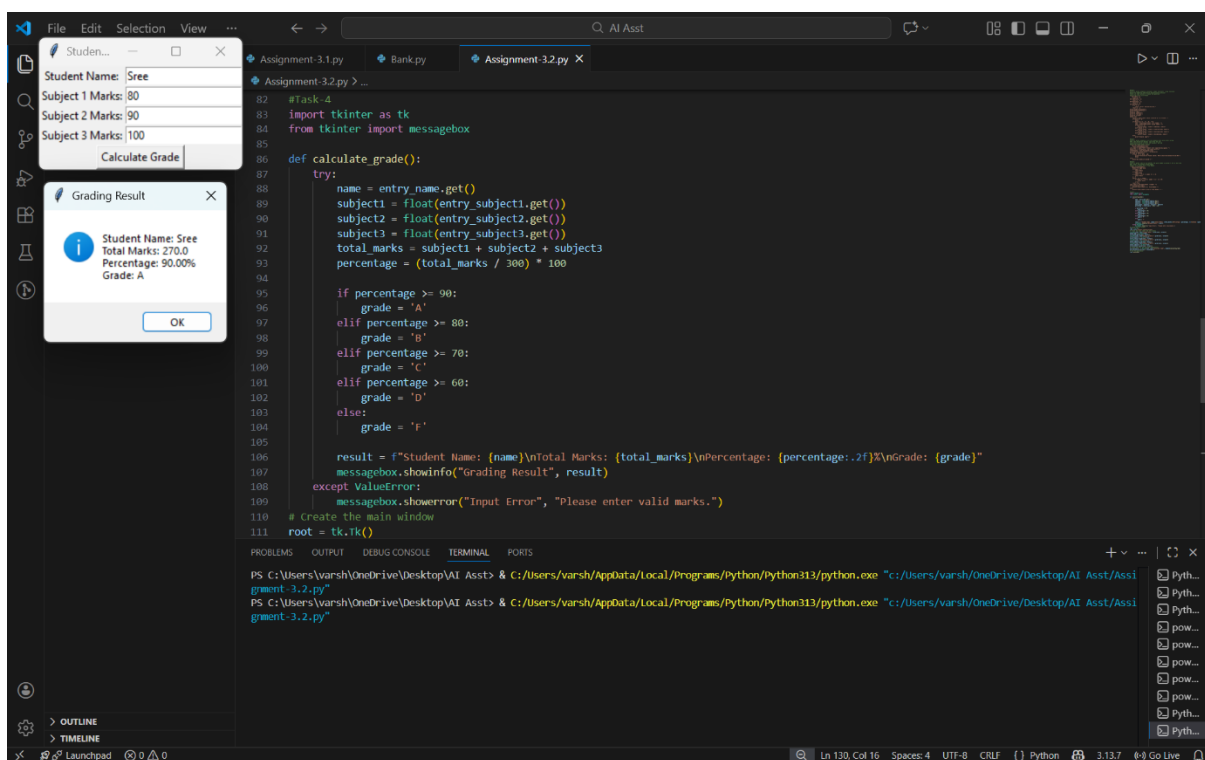
student grading system that calculates total marks, percentage, and grade based on user

input.

Expected Output-4

- Well-structured UI code with accurate calculations and clear output display.

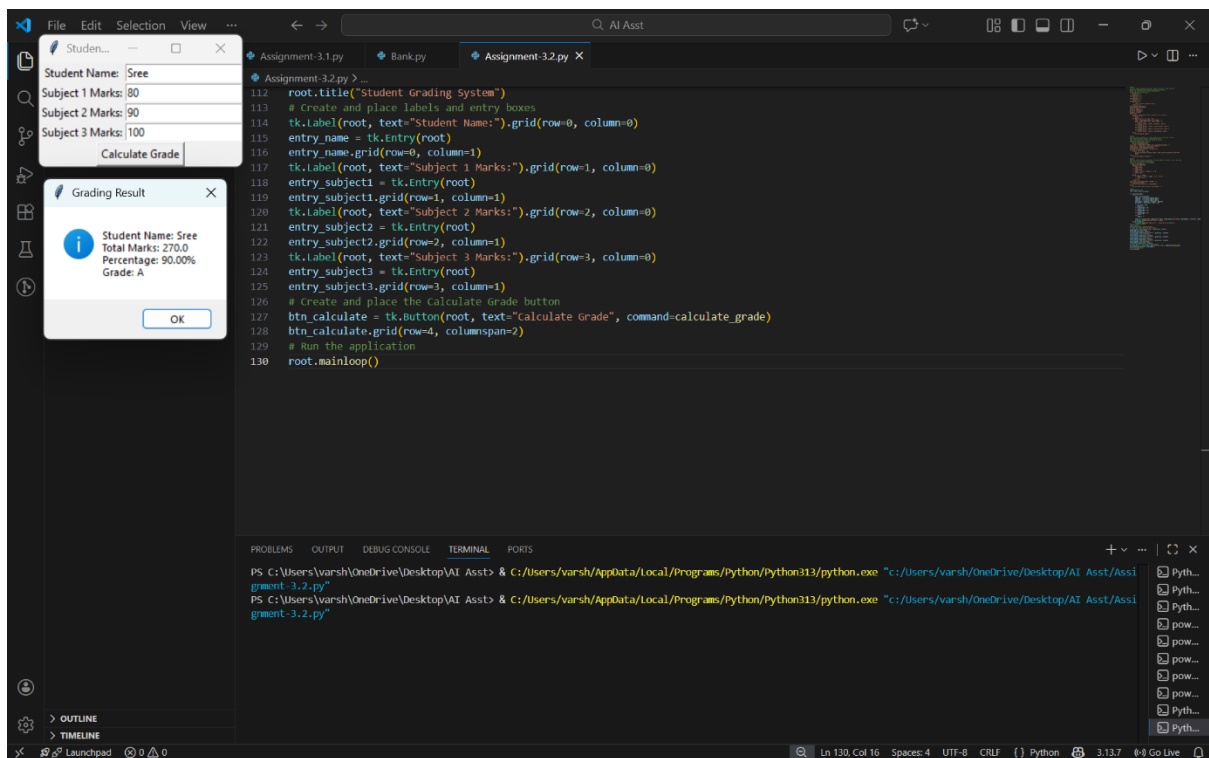
Code and Output:



The screenshot displays a Python IDE with a dark theme. The main editor window shows a Python script for a student grading system. The code uses Tkinter for the GUI and messagebox for displaying results and errors. It calculates the total marks, percentage, and assigns a grade based on the percentage.

```
82 #Task-4
83 import tkinter as tk
84 from tkinter import messagebox
85
86
87
88 def calculate_grade():
89     try:
90         name = entry_name.get()
91         subject1 = float(entry_subject1.get())
92         subject2 = float(entry_subject2.get())
93         subject3 = float(entry_subject3.get())
94         total_marks = subject1 + subject2 + subject3
95         percentage = (total_marks / 300) * 100
96
97         if percentage >= 90:
98             grade = 'A'
99         elif percentage >= 80:
100             grade = 'B'
101         elif percentage >= 70:
102             grade = 'C'
103         elif percentage >= 60:
104             grade = 'D'
105         else:
106             grade = 'F'
107
108         result = f"Student Name: {name}\nTotal Marks: {total_marks}\nPercentage: {percentage:.2f}%\nGrade: {grade}"
109         messagebox.showinfo("Grading Result", result)
110     except ValueError:
111         messagebox.showerror("Input Error", "Please enter valid marks.")
112
113 # Create the main window
114 root = tk.Tk()
```

On the left, there is a 'Student...' dialog box with input fields for 'Student Name' (Sree), 'Subject 1 Marks' (80), 'Subject 2 Marks' (90), and 'Subject 3 Marks' (100). Below these fields is a 'Calculate Grade' button. Below the dialog box is a 'Grading Result' dialog box showing the output: 'Student Name: Sree', 'Total Marks: 270.0', 'Percentage: 90.00%', and 'Grade: A'. At the bottom, the terminal window shows the command to run the script: `PS C:\Users\varsh\OneDrive\Desktop\AI Asst> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/varsh/OneDrive/Desktop/AI Asst/Assignment-3.2.py"`.



Analysis:

Prompt-Guided UI Design for Student Grading System

Prompt-guided design gives clear instructions for UI creation.

The prompt specifies inputs like student marks.

The system calculates total marks and percentage automatically.

It assigns grades based on rules given in the prompt.

Clear prompts help design a user-friendly interface.

Prompt guidance reduces errors and confusion in UI behavior.

Task Description-5

- Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear Instructions.

Expected Output-5

- Analysis of code quality and accuracy differences across multiple prompt variations.

Code and Output:

The screenshot shows a VS Code editor with a Python file named 'Assignment-3.2.py'. The code implements two functions: `km_to_miles` and `miles_to_km`, which convert between kilometers and miles using the conversion factor 0.621371. The script includes a menu for the user to choose the conversion direction and handles input validation. The terminal at the bottom shows the script being executed, with user input '1' for the first test case (60 km to miles) and '2' for the second test case (20 miles to km), resulting in the correct conversions.

```
132 #Task-5
133 #Write a python program to implement conversion of kilometer to miles and miles to kilometer
134 def km_to_miles(km):
135     return km * 0.621371
136 def miles_to_km(miles):
137     return miles / 0.621371
138 print("Conversion Options:")
139 print("1. Kilometer to Miles")
140 print("2. Miles to Kilometer")
141 choice = input("Enter choice (1/2): ")
142
143 if choice == '1':
144     km = float(input("Enter distance in kilometers: "))
145     miles = km_to_miles(km)
146     print(f"{km} kilometers is equal to {miles:.2f} miles.")
147 elif choice == '2':
148     miles = float(input("Enter distance in miles: "))
149     km = miles_to_km(miles)
150     print(f"{miles} miles is equal to {km:.2f} kilometers.")
151 else:
152     print("Invalid choice.")
```

2. Miles to Kilometer
Enter choice (1/2): 1
Enter distance in kilometers: 60
60.0 kilometers is equal to 37.28 miles.
PS C:\Users\varsh\OneDrive\Desktop\VAI Asst> & C:/Users/varsh/Appdata/Local/Programs/Python/python313/python.exe "c:/Users/varsh/OneDrive/Desktop/VAI Asst/Assignment-3.2.py"
Conversion Options:
1. Kilometer to Miles
2. Miles to kilometer
Enter choice (1/2): 2
Enter distance in miles: 20
20.0 miles is equal to 32.19 kilometers.
PS C:\Users\varsh\OneDrive\Desktop\VAI Asst>

Analysis:

Prompt Specificity in Unit Conversion Functions

Prompt specificity means giving clear and detailed instructions.

Initially, a vague prompt may cause incorrect or incomplete conversion.

Clear instructions specify conversion type (km to miles, miles to km).

The prompt defines input and output format clearly.

Specific prompts improve accuracy and correctness of the function.

This reduces errors and misunderstanding in unit conversion.