AI Assisted Coding

Assignment – 3.2

Name:N.VARSHITH

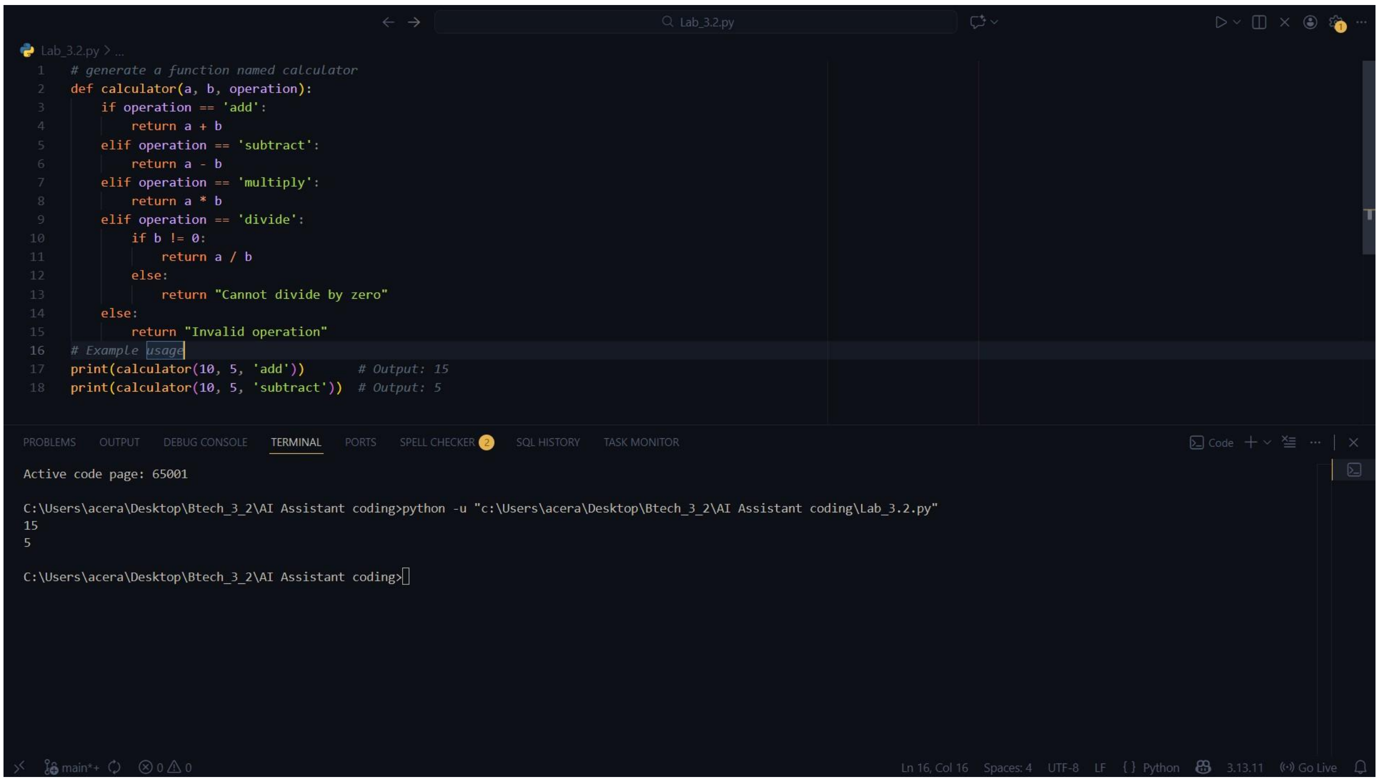
Batch: **21**

HtNo: **2303A51441**

Question 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.

Stage 1:

Code:

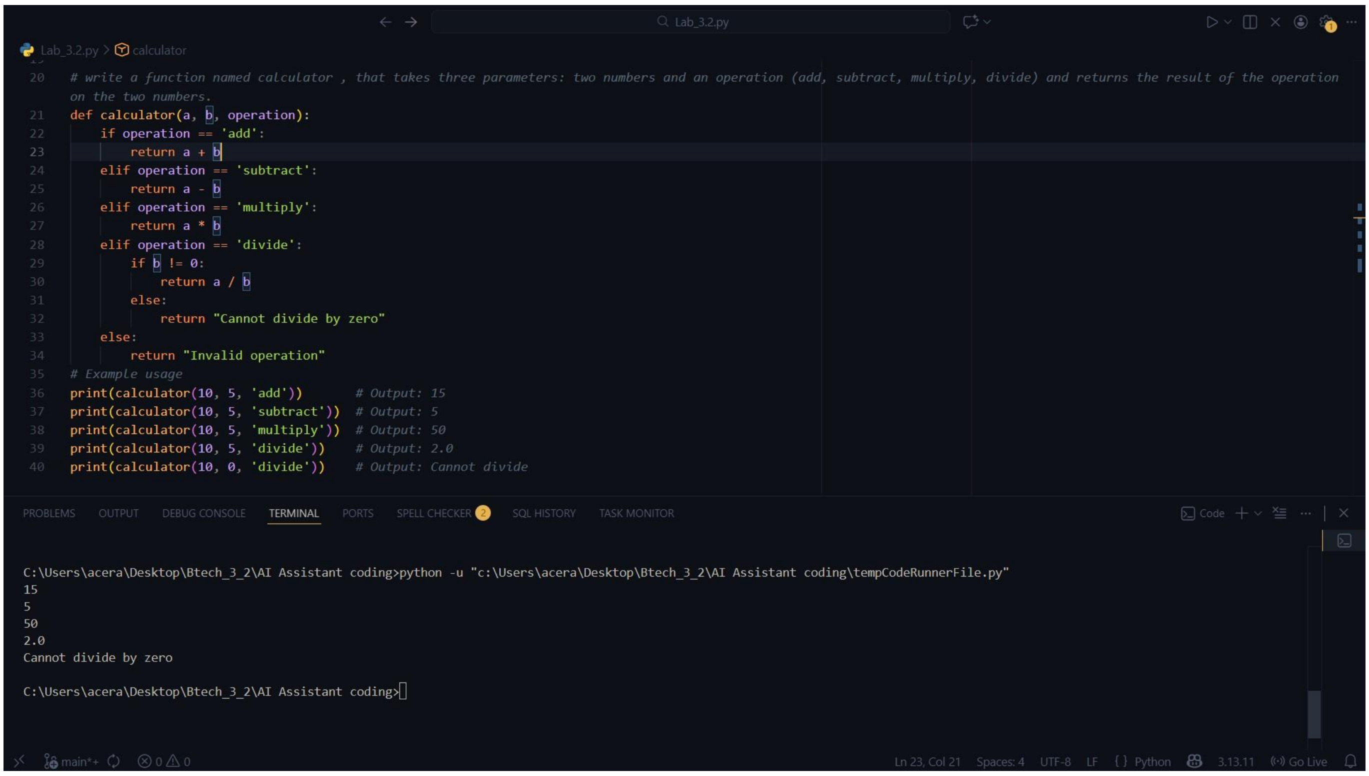


```python
# generate a function named calculator
def calculator(a, b, operation):
    if operation == 'add':
        return a + b
    elif operation == 'subtract':
        return a - b
    elif operation == 'multiply':
        return a * b
    elif operation == 'divide':
        if b != 0:
            return a / b
        else:
            return "Cannot divide by zero"
    else:
        return "Invalid operation"
# Example usage
print(calculator(10, 5, 'add'))       # Output: 15
print(calculator(10, 5, 'subtract'))  # Output: 5
```

```
Active code page: 65001

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\Lab_3.2.py"
15
5

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>
```

Stage 2:

Code:



Lab_3.2.py > calculator
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER SQL HISTORY TASK MONITOR

```python
# write a function named calculator , that takes three parameters: two numbers and an operation (add, subtract, multiply, divide) and returns the result of the operation on the two numbers.
def calculator(a, b, operation):
    if operation == 'add':
        return a + b
    elif operation == 'subtract':
        return a - b
    elif operation == 'multiply':
        return a * b
    elif operation == 'divide':
        if b != 0:
            return a / b
        else:
            return "Cannot divide by zero"
    else:
        return "Invalid operation"
# Example usage
print(calculator(10, 5, 'add'))       # Output: 15
print(calculator(10, 5, 'subtract'))  # Output: 5
print(calculator(10, 5, 'multiply'))  # Output: 50
print(calculator(10, 5, 'divide'))    # Output: 2.0
print(calculator(10, 0, 'divide'))    # Output: Cannot divide
```

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
15
5
50
2.0
Cannot divide by zero

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>
```

Final Observation:

At first, when only the function name was given, the AI generated a very basic and incomplete calculator function with little or no logic. After adding comments, the AI started including parameters and arithmetic operations. When usage examples were finally added, the AI produced a complete and well-structured calculator program with proper conditions and error handling. This clearly shows that progressive prompting improves both the logic and structure of the generated code.

Question 2: 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.

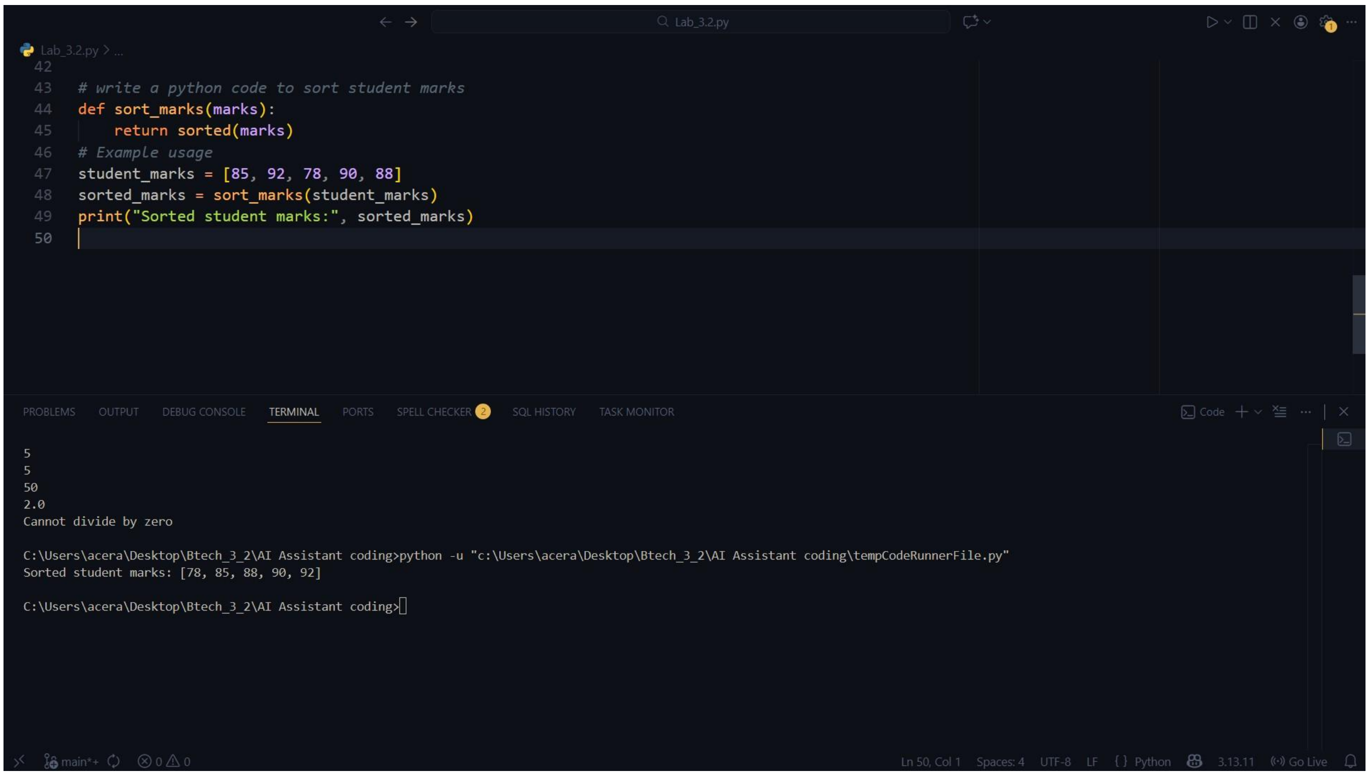
Expected Output-2

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

Stage 1:

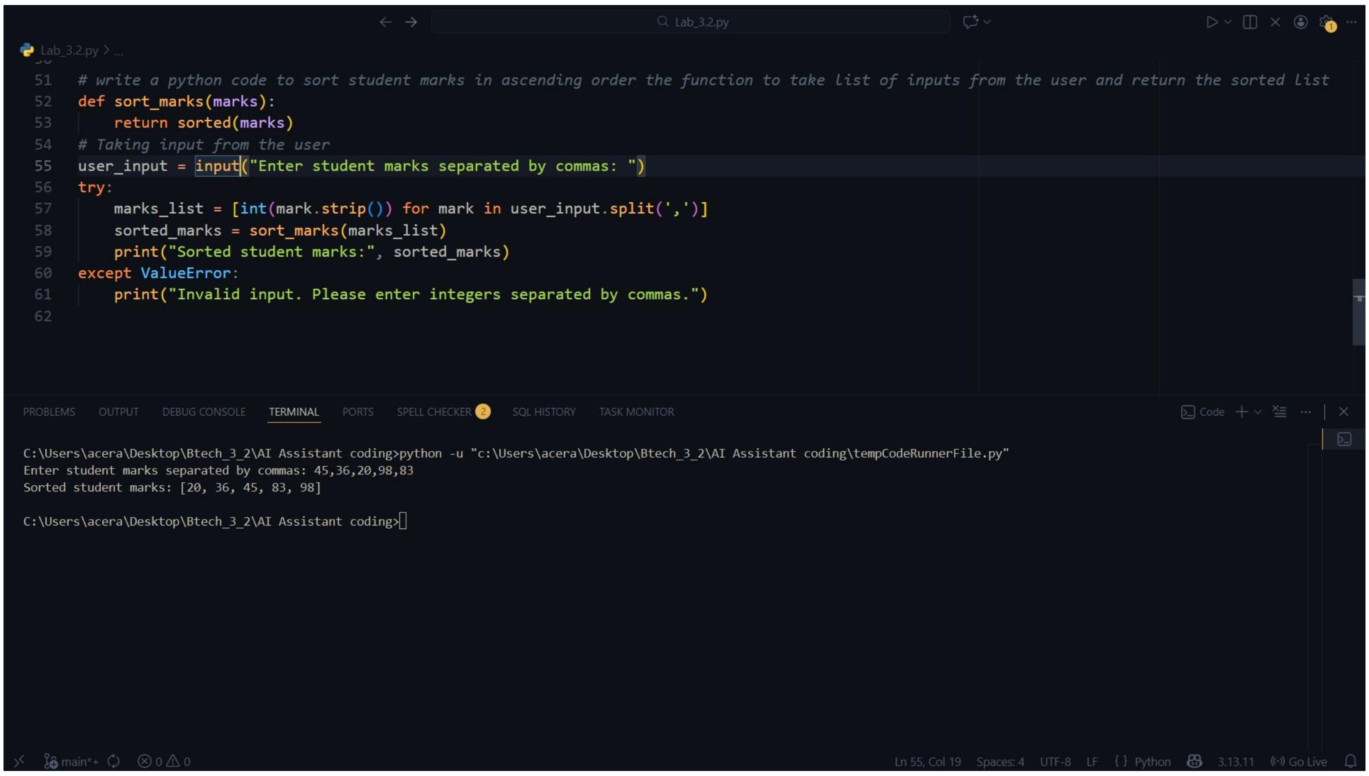
Code and

output:



```python
# write a python code to sort student marks
def sort_marks(marks):
    return sorted(marks)
# Example usage
student_marks = [85, 92, 78, 90, 88]
sorted_marks = sort_marks(student_marks)
print("Sorted student marks:", sorted_marks)
```

```
5
5
50
2.0
Cannot divide by zero

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
Sorted student marks: [78, 85, 88, 90, 92]

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>
```

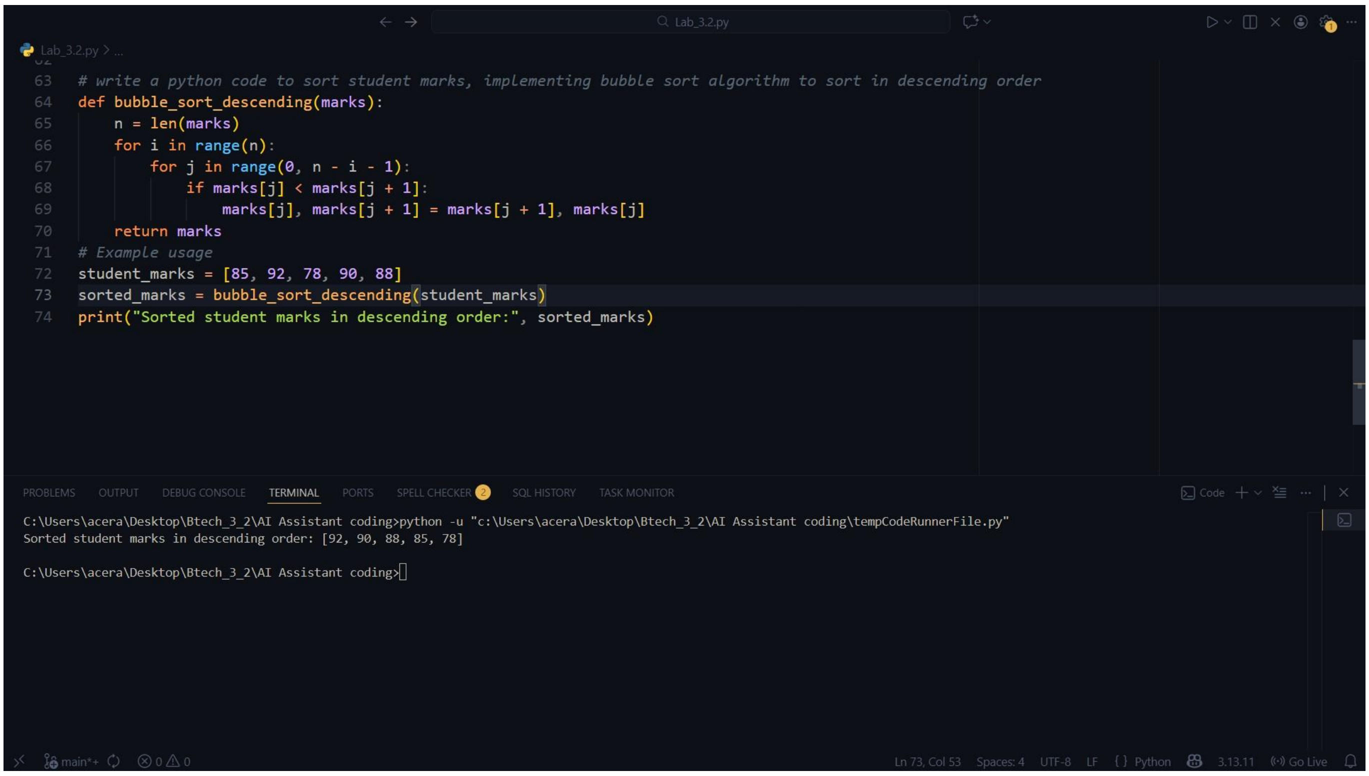


```python
# write a python code to sort student marks in ascending order the function to take list of inputs from the user and return the sorted list
def sort_marks(marks):
    return sorted(marks)
# Taking input from the user
user_input = input("Enter student marks separated by commas: ")
try:
    marks_list = [int(mark.strip()) for mark in user_input.split(',')]
    sorted_marks = sort_marks(marks_list)
    print("Sorted student marks:", sorted_marks)
except ValueError:
    print("Invalid input. Please enter integers separated by commas.")
```

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
Enter student marks separated by commas: 45,36,20,98,83
Sorted student marks: [20, 36, 45, 83, 98]

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>
```

Stage 3:

Code:



```python
# write a python code to sort student marks, implementing bubble sort algorithm to sort in descending order
def bubble_sort_descending(marks):
    n = len(marks)
    for i in range(n):
        for j in range(0, n - i - 1):
            if marks[j] < marks[j + 1]:
                marks[j], marks[j + 1] = marks[j + 1], marks[j]
    return marks
# Example usage
student_marks = [85, 92, 78, 90, 88]
sorted_marks = bubble_sort_descending(student_marks)
print("Sorted student marks in descending order:", sorted_marks)
```

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
Sorted student marks in descending order: [92, 90, 88, 85, 78]

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>
```

Final Observation:

With a vague prompt, the AI produced a simple sorting solution without clear direction or constraints. After refining the prompt to specify sorting order, the output became more accurate and meaningful. When clear constraints and examples were added, the AI generated a more structured and efficient sorting function. This demonstrates that refining prompts helps the AI move from ambiguous logic to a correct and reliable implementation.

Question 3: Task Description-3

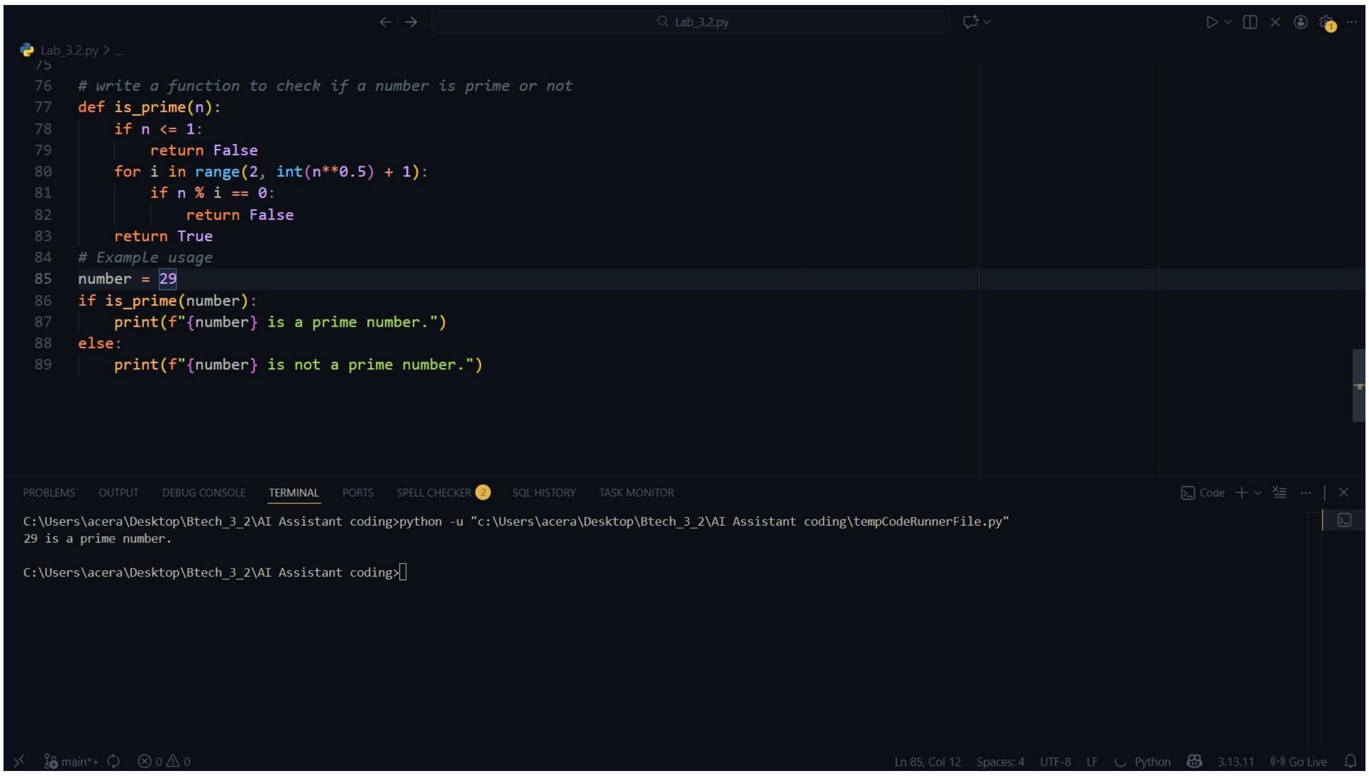
• Few-Shot Prompting for Prime Number Validation: Provide multiple input-output examples for a function that checks whether a number is prime. Observe how few-shot prompting improves correctness.

Expected Output-3

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

Stage 1:

Code:



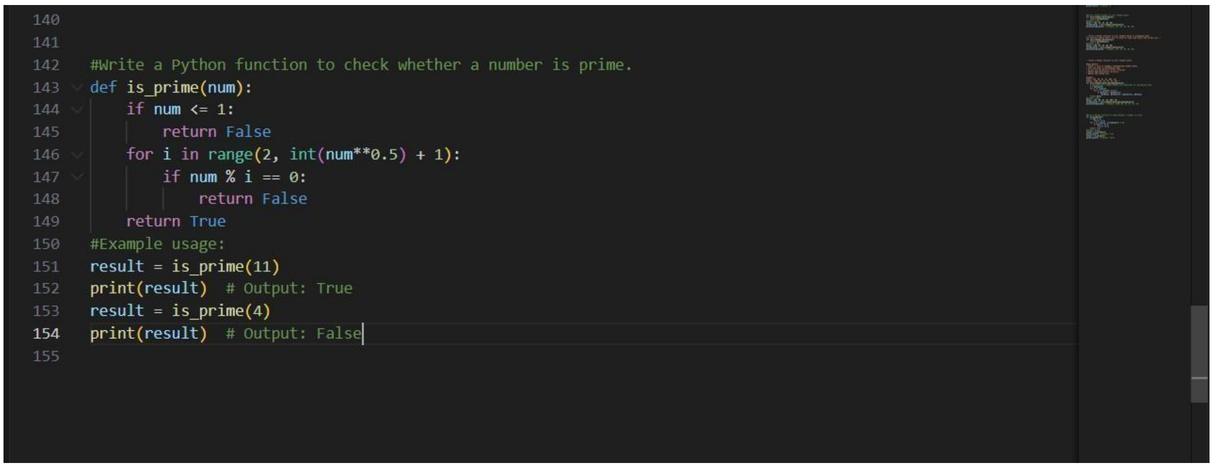
```python
# write a function to check if a number is prime or not
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
# Example usage
number = 29
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
29 is a prime number.

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>
```

Lab_3.2.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 2 SQL HISTORY TASK MONITOR
main*+ 0 0
Ln 85, Col 12 Spaces: 4 UTF-8 LF Python 3.13.11 Go Live

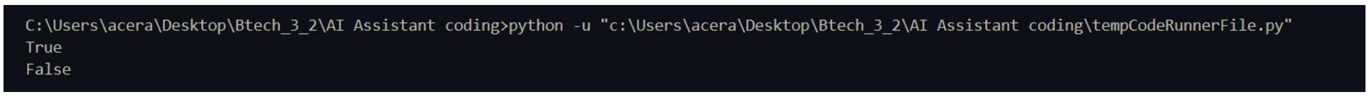
Stage 2:

Code:



```python
#Write a Python function to check whether a number is prime.
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True
#Example usage:
result = is_prime(11)
print(result)  # Output: True
result = is_prime(4)
print(result)  # Output: False
```

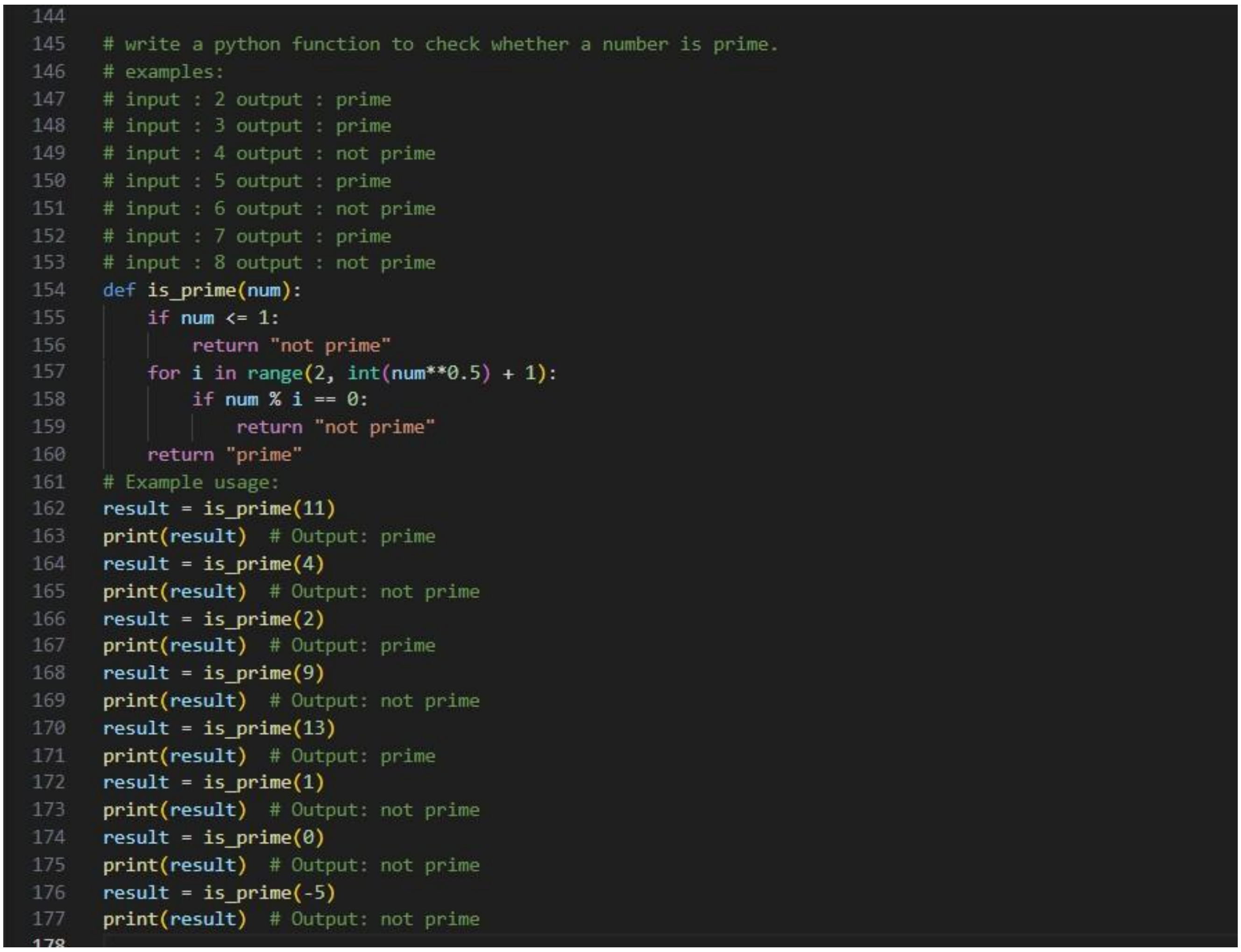
Output:



```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
True
False
```

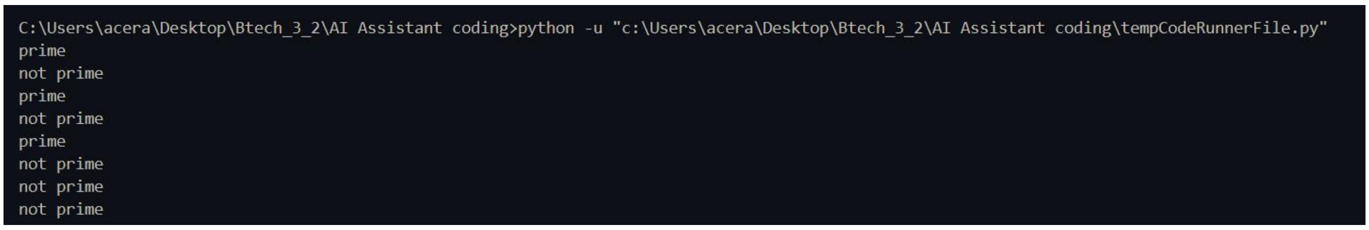
Stage 3:

Code:



```python
# write a python function to check whether a number is prime.
# examples:
# input : 2 output : prime
# input : 3 output : prime
# input : 4 output : not prime
# input : 5 output : prime
# input : 6 output : not prime
# input : 7 output : prime
# input : 8 output : not prime
def is_prime(num):
    if num <= 1:
        return "not prime"
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return "not prime"
    return "prime"
# Example usage:
result = is_prime(11)
print(result)  # Output: prime
result = is_prime(4)
print(result)  # Output: not prime
result = is_prime(2)
print(result)  # Output: prime
result = is_prime(9)
print(result)  # Output: not prime
result = is_prime(13)
print(result)  # Output: prime
result = is_prime(1)
print(result)  # Output: not prime
result = is_prime(0)
print(result)  # Output: not prime
result = is_prime(-5)
print(result)  # Output: not prime
```

Output:



```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
prime
not prime
prime
not prime
prime
not prime
not prime
not prime
```

Final Observation:

In the initial prompt without examples, the AI generated a basic prime-checking function that could miss important edge cases. When one example was provided, the result improved slightly. After giving multiple input-output examples (few-shot prompting), the AI clearly handled cases like 0, 1, and negative numbers and produced a more accurate and robust prime-checking function. This shows that few-shot prompting improves correctness and edge-case handling.

Question 4: Task Description-4

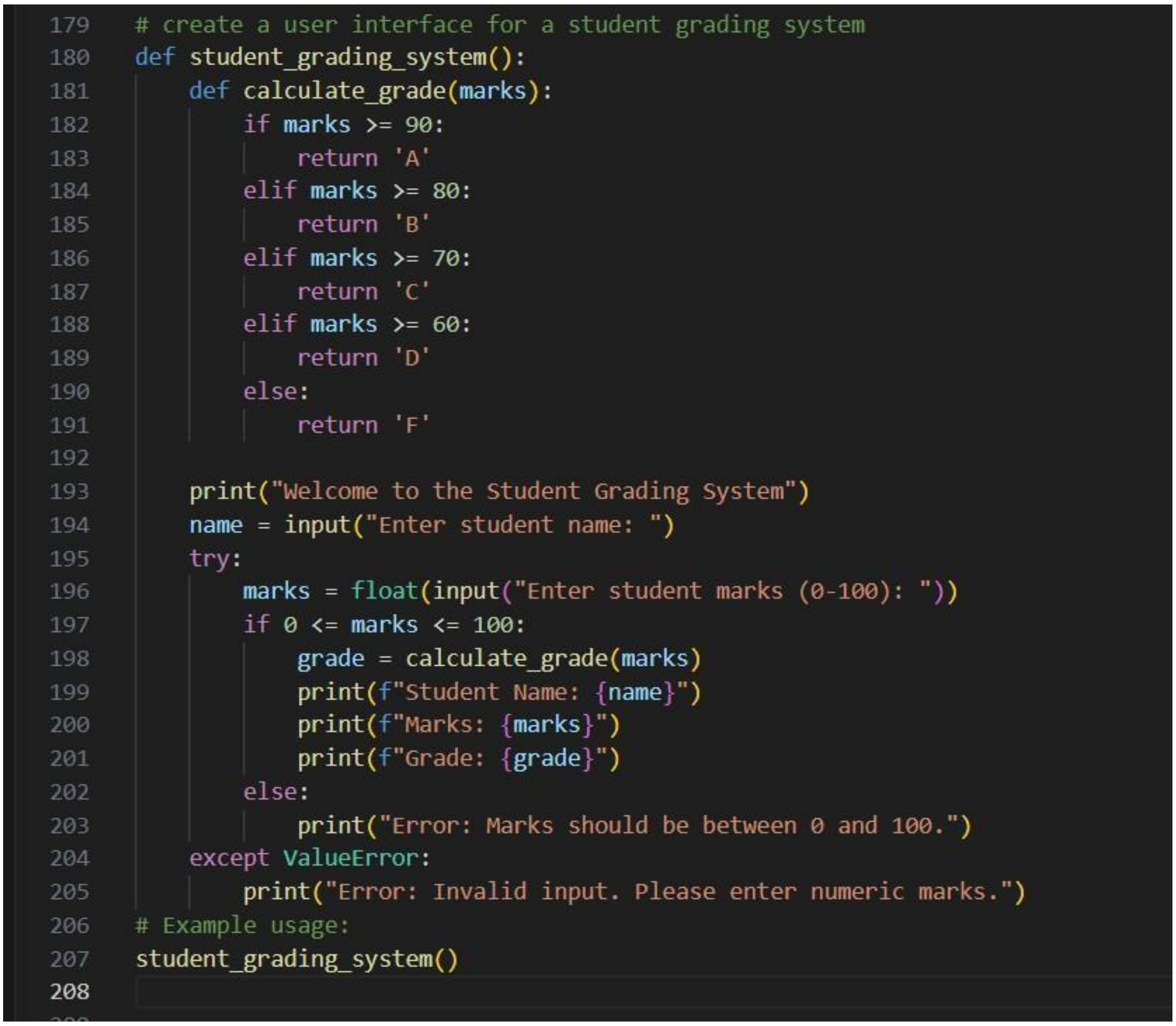
- Prompt-Guided UI Design for Student Grading System: Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

Expected Output-4

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

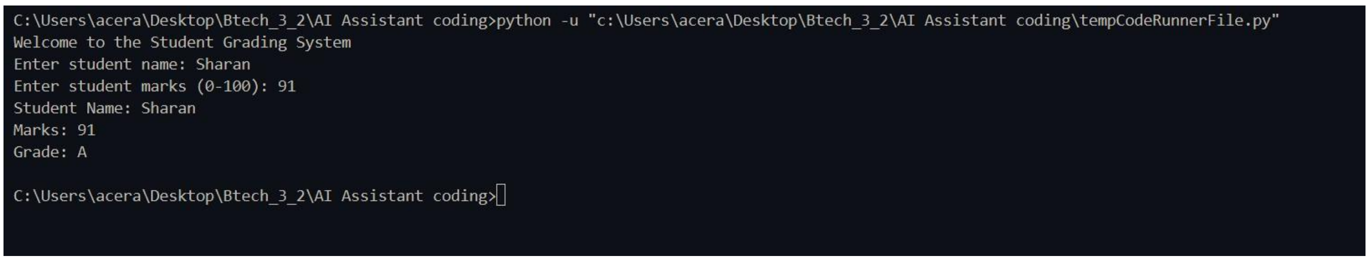
Stage 1:

Code:



```python
# create a user interface for a student grading system
def student_grading_system():
    def calculate_grade(marks):
        if marks >= 90:
            return 'A'
        elif marks >= 80:
            return 'B'
        elif marks >= 70:
            return 'C'
        elif marks >= 60:
            return 'D'
        else:
            return 'F'

    print("Welcome to the Student Grading System")
    name = input("Enter student name: ")
    try:
        marks = float(input("Enter student marks (0-100): "))
        if 0 <= marks <= 100:
            grade = calculate_grade(marks)
            print(f"Student Name: {name}")
            print(f"Marks: {marks}")
            print(f"Grade: {grade}")
        else:
            print("Error: Marks should be between 0 and 100.")
    except ValueError:
        print("Error: Invalid input. Please enter numeric marks.")
# Example usage:
student_grading_system()

```

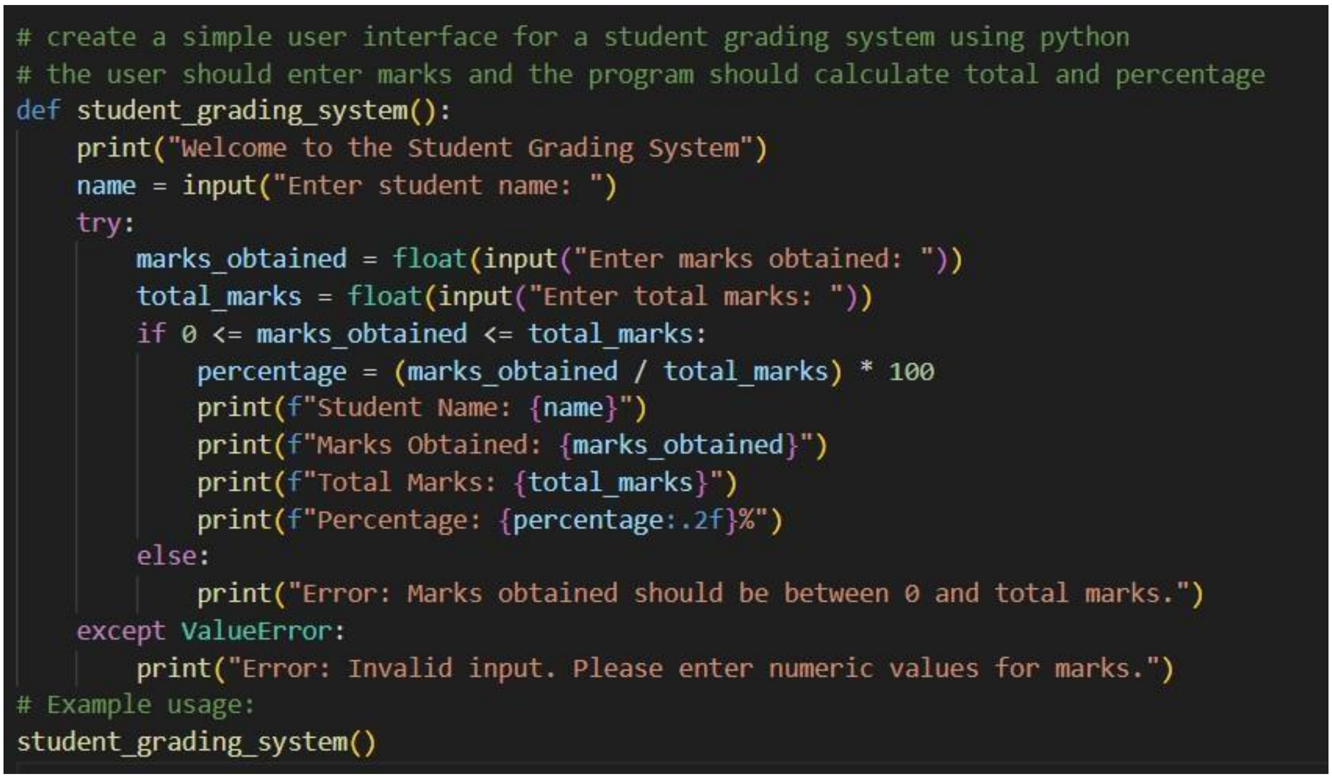
Output:



```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
Welcome to the Student Grading System
Enter student name: Sharan
Enter student marks (0-100): 91
Student Name: Sharan
Marks: 91
Grade: A

C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>
```

Stage 2:

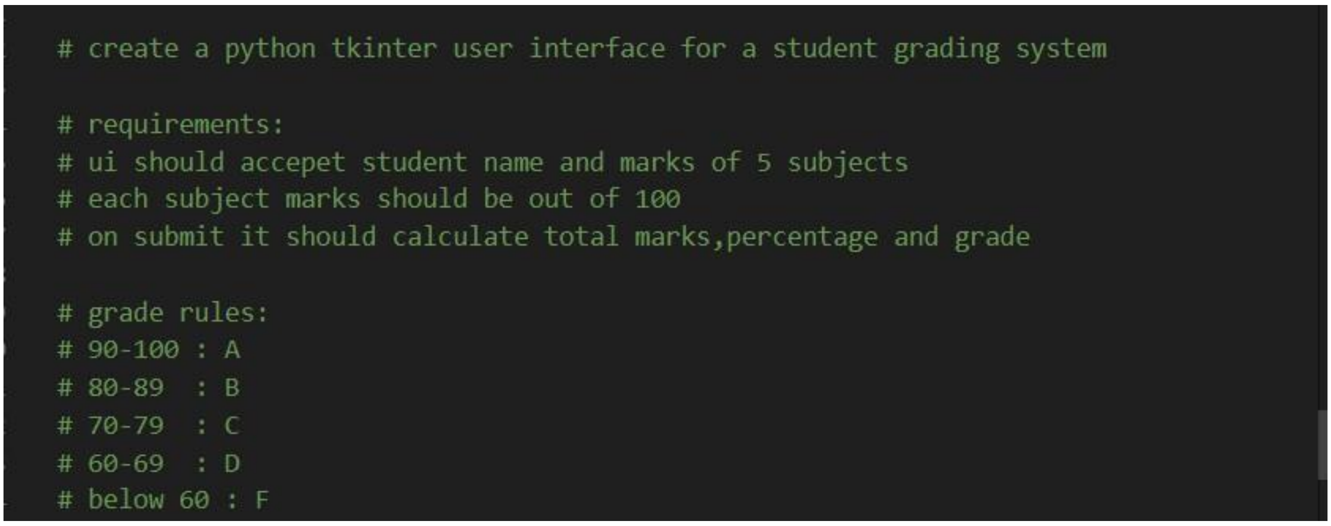
Code:



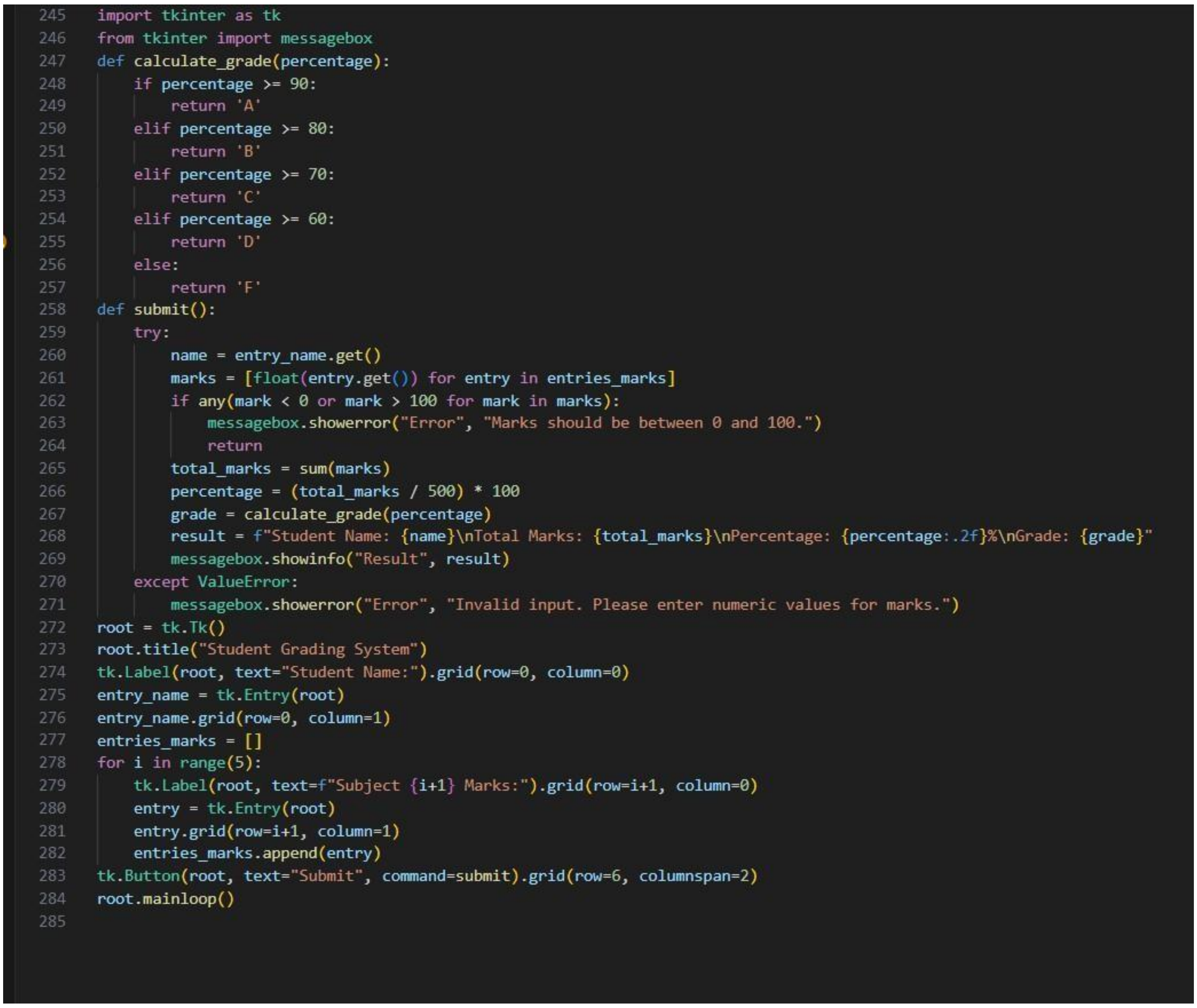
```python
# create a simple user interface for a student grading system using python
# the user should enter marks and the program should calculate total and percentage
def student_grading_system():
    print("Welcome to the Student Grading System")
    name = input("Enter student name: ")
    try:
        marks_obtained = float(input("Enter marks obtained: "))
        total_marks = float(input("Enter total marks: "))
        if 0 <= marks_obtained <= total_marks:
            percentage = (marks_obtained / total_marks) * 100
            print(f"Student Name: {name}")
            print(f"Marks Obtained: {marks_obtained}")
            print(f"Total Marks: {total_marks}")
            print(f"Percentage: {percentage:.2f}%")
        else:
            print("Error: Marks obtained should be between 0 and total marks.")
    except ValueError:
        print("Error: Invalid input. Please enter numeric values for marks.")
# Example usage:
student_grading_system()
```

Stage 3:

Code:

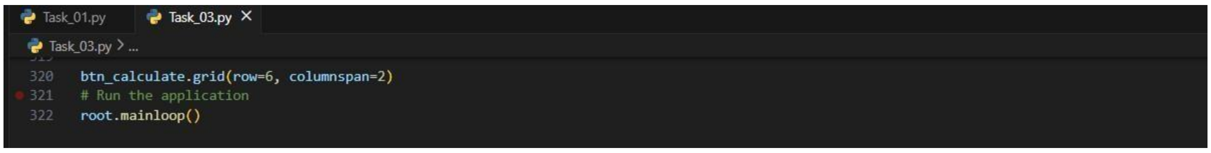


```python
# create a python tkinter user interface for a student grading system

# requirements:
# ui should accepet student name and marks of 5 subjects
# each subject marks should be out of 100
# on submit it should calculate total marks,percentage and grade

# grade rules:
# 90-100 : A
# 80-89  : B
# 70-79  : C
# 60-69  : D
# below 60 : F
```

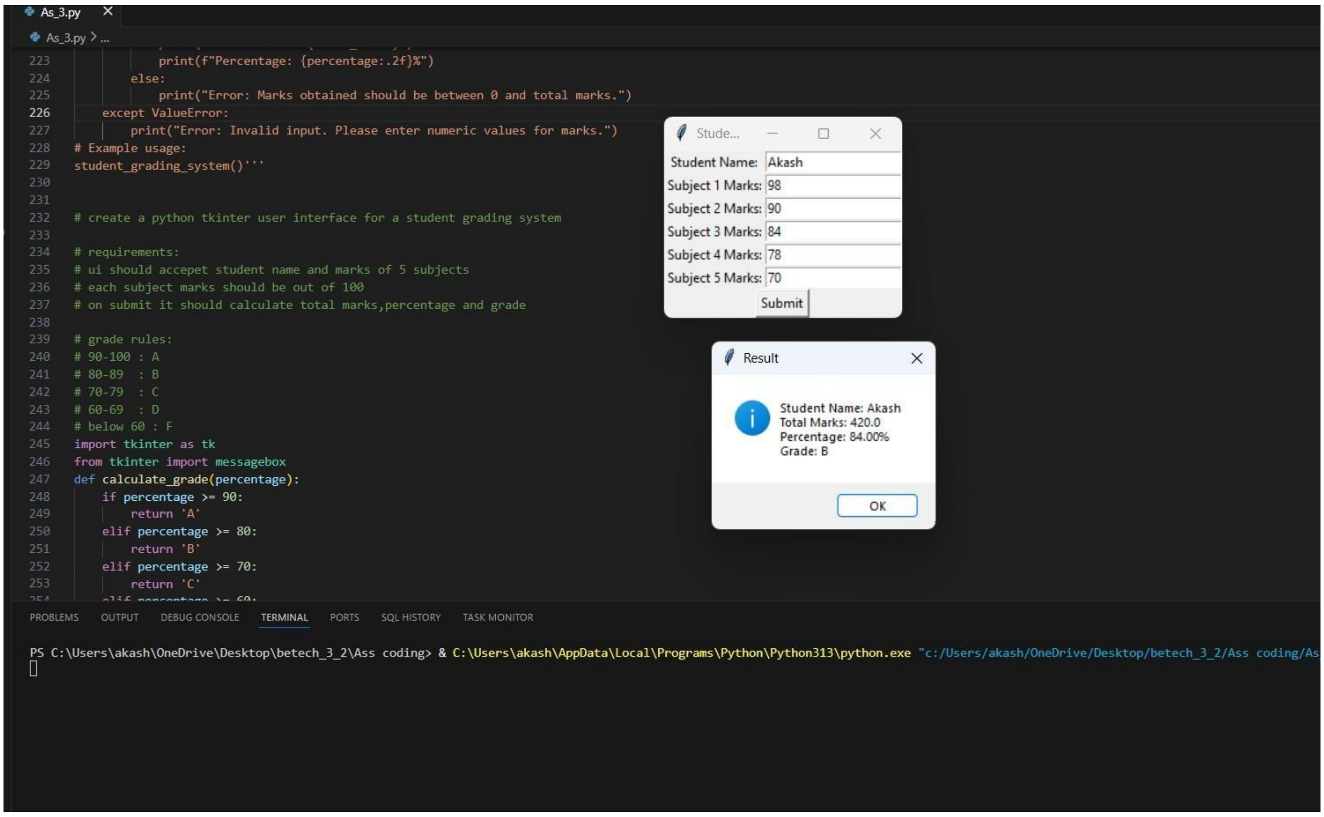


```python
import tkinter as tk
from tkinter import messagebox
def calculate_grade(percentage):
    if percentage >= 90:
        return 'A'
    elif percentage >= 80:
        return 'B'
    elif percentage >= 70:
        return 'C'
    elif percentage >= 60:
        return 'D'
    else:
        return 'F'
def submit():
    try:
        name = entry_name.get()
        marks = [float(entry.get()) for entry in entries_marks]
        if any(mark < 0 or mark > 100 for mark in marks):
            messagebox.showerror("Error", "Marks should be between 0 and 100.")
            return
        total_marks = sum(marks)
        percentage = (total_marks / 500) * 100
        grade = calculate_grade(percentage)
        result = f"Student Name: {name}\nTotal Marks: {total_marks}\nPercentage: {percentage:.2f}%\nGrade: {grade}"
        messagebox.showinfo("Result", result)
    except ValueError:
        messagebox.showerror("Error", "Invalid input. Please enter numeric values for marks.")
root = tk.Tk()
root.title("Student Grading System")
tk.Label(root, text="Student Name:").grid(row=0, column=0)
entry_name = tk.Entry(root)
entry_name.grid(row=0, column=1)
entries_marks = []
for i in range(5):
    tk.Label(root, text=f"Subject {i+1} Marks:").grid(row=i+1, column=0)
    entry = tk.Entry(root)
    entry.grid(row=i+1, column=1)
    entries_marks.append(entry)
tk.Button(root, text="Submit", command=submit).grid(row=6, columnspan=2)
root.mainloop()
```

Output:



```python
btn_calculate.grid(row=6, columnspan=2)
# Run the application
root.mainloop()
```



```
                print(f"Percentage: {percentage:.2f}%")
            else:
                print("Error: Marks obtained should be between 0 and total marks.")
        except ValueError:
            print("Error: Invalid input. Please enter numeric values for marks.")
    # Example usage:
    student_grading_system()'''


    # create a python tkinter user interface for a student grading system

    # requirements:
    # ui should accepet student name and marks of 5 subjects
    # each subject marks should be out of 100
    # on submit it should calculate total marks,percentage and grade

    # grade rules:
    # 90-100 : A
    # 80-89  : B
    # 70-79  : C
    # 60-69  : D
    # below 60 : F
    import tkinter as tk
    from tkinter import messagebox
    def calculate_grade(percentage):
        if percentage >= 90:
            return 'A'
        elif percentage >= 80:
            return 'B'
        elif percentage >= 70:
            return 'C'
```

```
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\Ass coding> & C:\Users\akash\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/akash/OneDrive/Desktop/betech_3_2/Ass coding/As
```

Student Name: Akash
Subject 1 Marks: 98
Subject 2 Marks: 90
Subject 3 Marks: 84
Subject 4 Marks: 78
Subject 5 Marks: 70

```
Student Name: Akash
Total Marks: 420.0
Percentage: 84.00%
Grade: B
```

Final Observation:

With a vague UI prompt, the AI produced only a simple or unclear interface idea. As the prompt was refined to include calculation requirements, the UI output became more meaningful. When full instructions were given (inputs, calculations, grade rules, and display), the AI generated a well-structured user interface with correct total, percentage, and grade calculation along with clear result display. This shows that prompt guidance greatly improves UI structure and usability.

Question 5: Task Description-5

• Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit

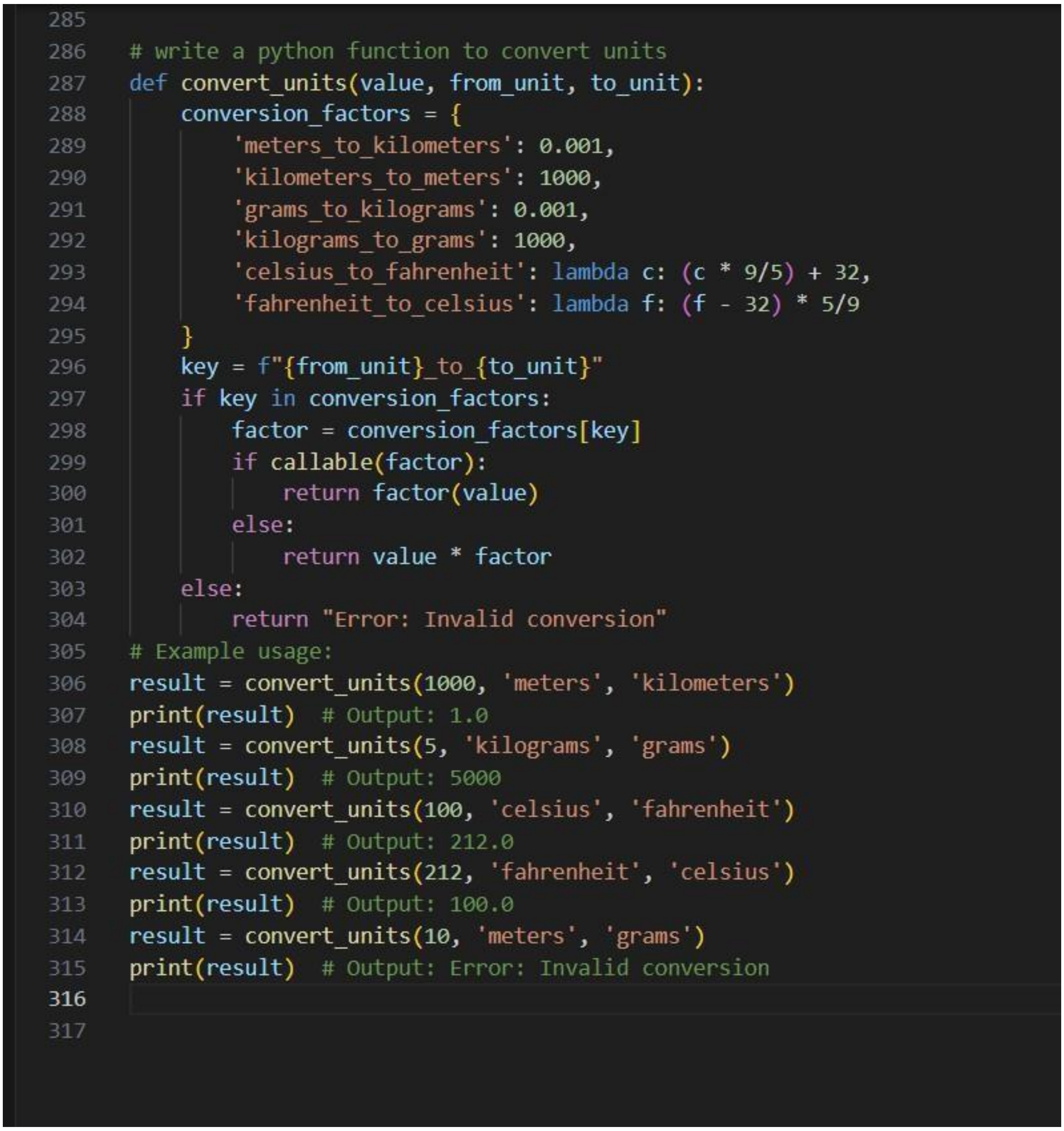
Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear Instructions.

Expected Output-5

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

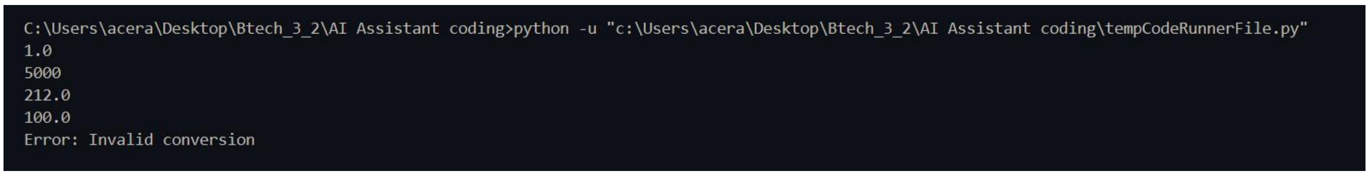
Stage 1:

Code:



```python

# write a python function to convert units
def convert_units(value, from_unit, to_unit):
    conversion_factors = {
        'meters_to_kilometers': 0.001,
        'kilometers_to_meters': 1000,
        'grams_to_kilograms': 0.001,
        'kilograms_to_grams': 1000,
        'celsius_to_fahrenheit': lambda c: (c * 9/5) + 32,
        'fahrenheit_to_celsius': lambda f: (f - 32) * 5/9
    }
    key = f"{from_unit}_to_{to_unit}"
    if key in conversion_factors:
        factor = conversion_factors[key]
        if callable(factor):
            return factor(value)
        else:
            return value * factor
    else:
        return "Error: Invalid conversion"
# Example usage:
result = convert_units(1000, 'meters', 'kilometers')
print(result)  # Output: 1.0
result = convert_units(5, 'kilograms', 'grams')
print(result)  # Output: 5000
result = convert_units(100, 'celsius', 'fahrenheit')
print(result)  # Output: 212.0
result = convert_units(212, 'fahrenheit', 'celsius')
print(result)  # Output: 100.0
result = convert_units(10, 'meters', 'grams')
print(result)  # Output: Error: Invalid conversion

```

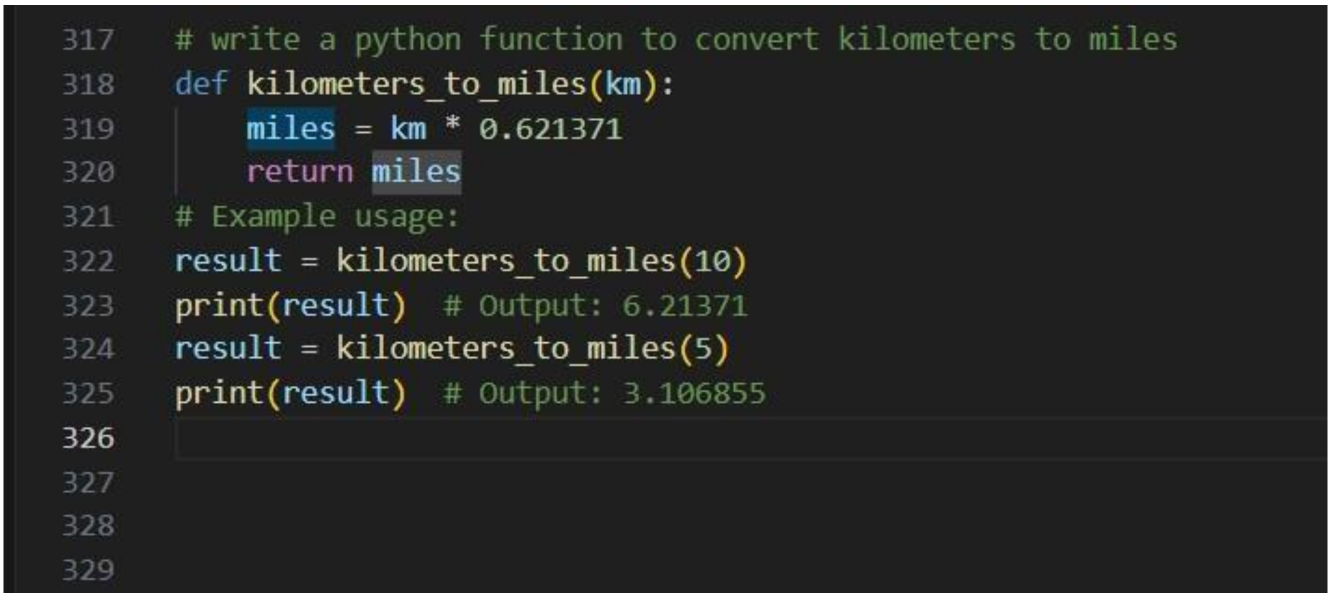
Output:



```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
1.0
5000
212.0
100.0
Error: Invalid conversion
```

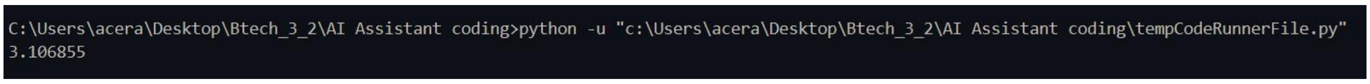
Stage 2:

Code:



```python
# write a python function to convert kilometers to miles
def kilometers_to_miles(km):
    miles = km * 0.621371
    return miles
# Example usage:
result = kilometers_to_miles(10)
print(result)  # Output: 6.21371
result = kilometers_to_miles(5)
print(result)  # Output: 3.106855
```

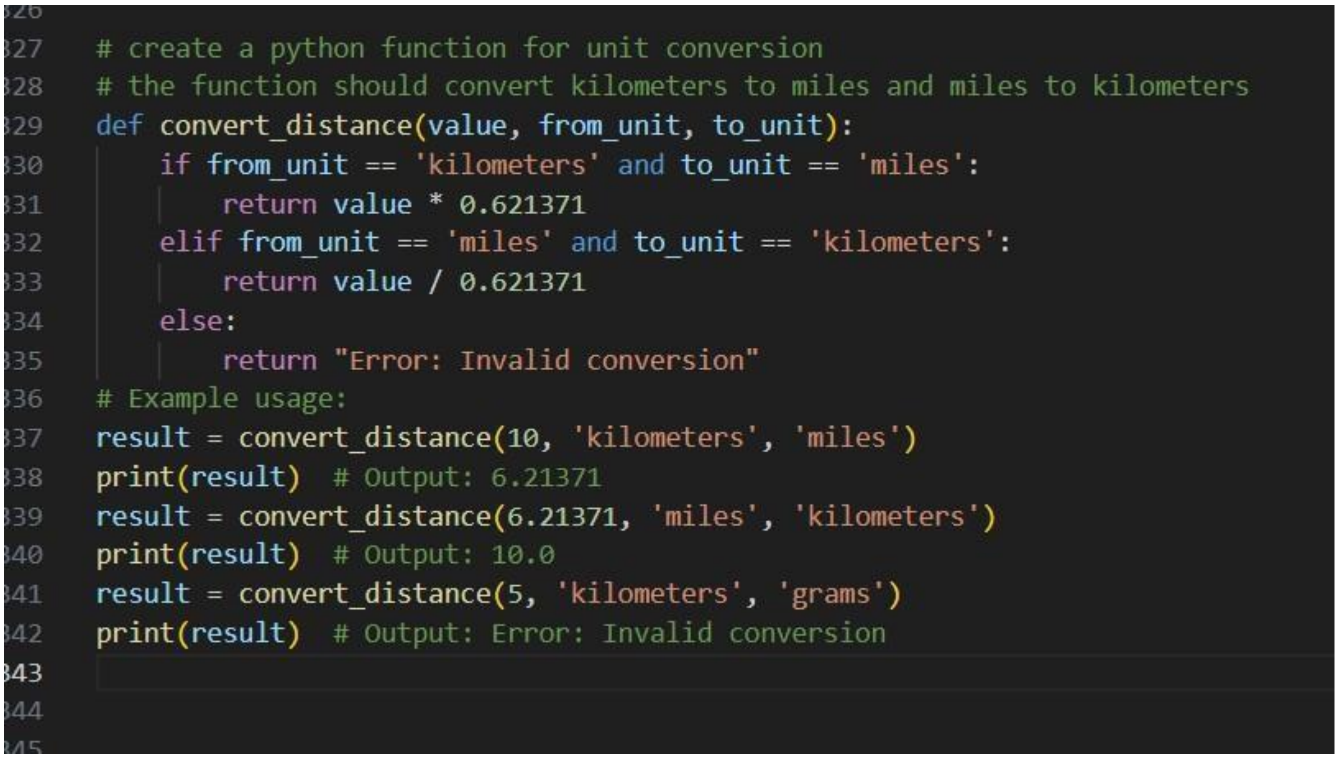
Output:



```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
3.106855
```

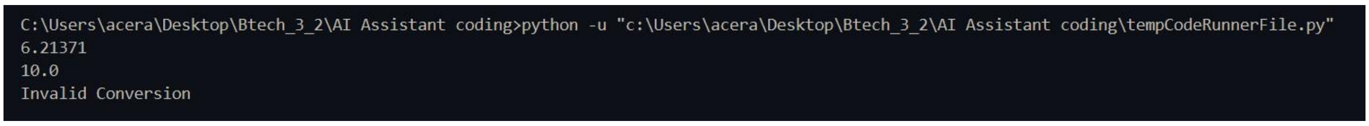
Stage 3:

Code:



```python
# create a python function for unit conversion
# the function should convert kilometers to miles and miles to kilometers
def convert_distance(value, from_unit, to_unit):
    if from_unit == 'kilometers' and to_unit == 'miles':
        return value * 0.621371
    elif from_unit == 'miles' and to_unit == 'kilometers':
        return value / 0.621371
    else:
        return "Error: Invalid conversion"
# Example usage:
result = convert_distance(10, 'kilometers', 'miles')
print(result)  # Output: 6.21371
result = convert_distance(6.21371, 'miles', 'kilometers')
print(result)  # Output: 10.0
result = convert_distance(5, 'kilometers', 'grams')
print(result)  # Output: Error: Invalid conversion
```

Output:



```
C:\Users\acera\Desktop\Btech_3_2\AI Assistant coding>python -u "c:\Users\acera\Desktop\Btech_3_2\AI Assistant coding\tempCodeRunnerFile.py"
6.21371
10.0
Invalid Conversion
```

Final Observation:

When a vague prompt was used, the AI generated unclear or very general conversion code. After specifying the type of conversion, the AI produced a basic one-way converter. When detailed instructions, formulas, and validation rules were added, the AI generated an accurate, well-structured, and reusable unit conversion function. This proves that higher prompt specificity leads to better code quality, accuracy, and reliability.