

Assignment-3.2

Name:M Varshith Reddy

HTNO:2303A52087

Batch:40

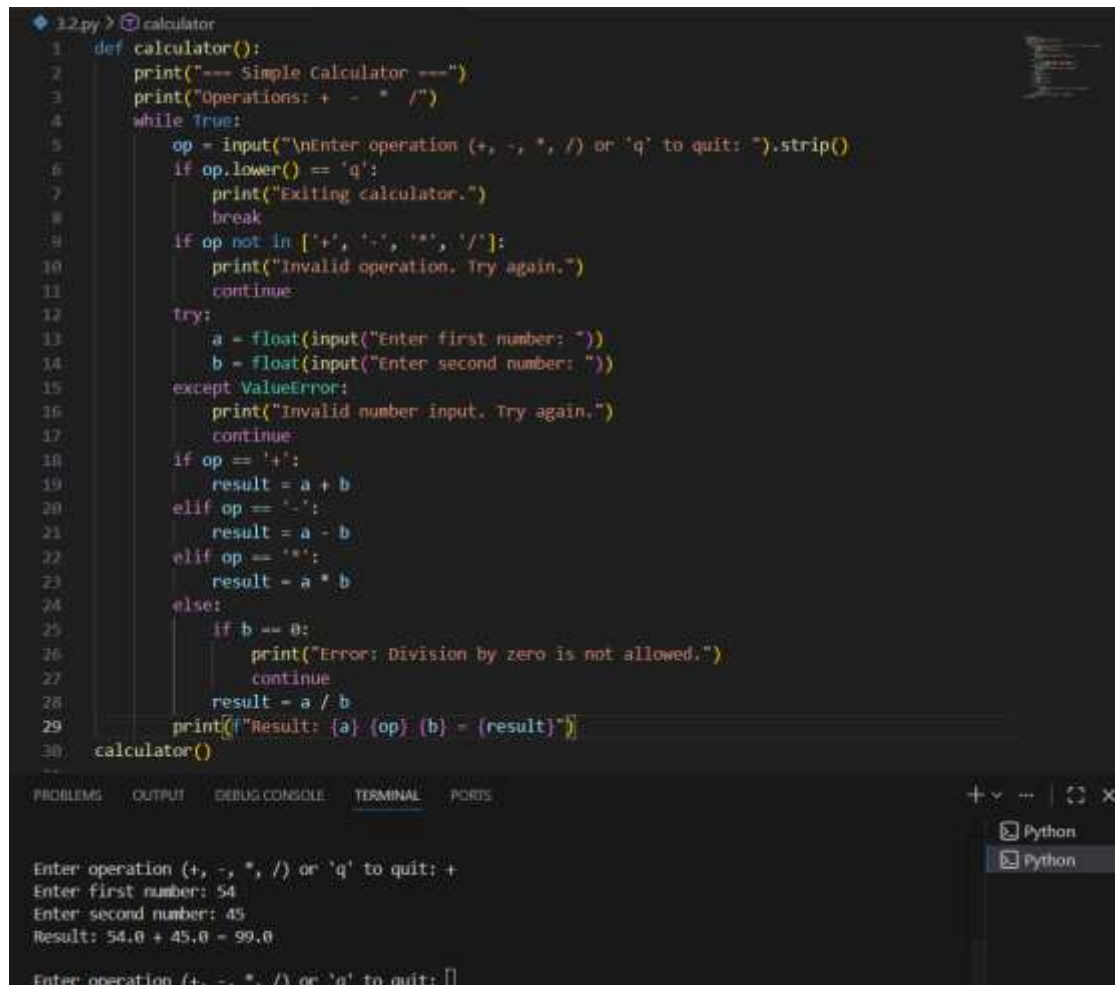
Task Description-1

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:



```
12.py > calculator
1 def calculator():
2     print("=== Simple Calculator ===")
3     print("Operations: + - * /")
4     while True:
5         op = input("\nEnter operation (+, -, *, /) or 'q' to quit: ").strip()
6         if op.lower() == 'q':
7             print("Exiting calculator.")
8             break
9         if op not in ['+', '-', '*', '/']:
10            print("Invalid operation. Try again.")
11            continue
12        try:
13            a = float(input("Enter first number: "))
14            b = float(input("Enter second number: "))
15        except ValueError:
16            print("Invalid number input. Try again.")
17            continue
18        if op == '+':
19            result = a + b
20        elif op == '-':
21            result = a - b
22        elif op == '*':
23            result = a * b
24        else:
25            if b == 0:
26                print("Error: Division by zero is not allowed.")
27                continue
28            result = a / b
29        print(f"Result: {a} {op} {b} = {result}")
30    calculator()
```

Enter operation (+, -, *, /) or 'q' to quit: +
Enter first number: 54
Enter second number: 45
Result: 54.0 + 45.0 = 99.0
Enter operation (+, -, *, /) or 'q' to quit: □

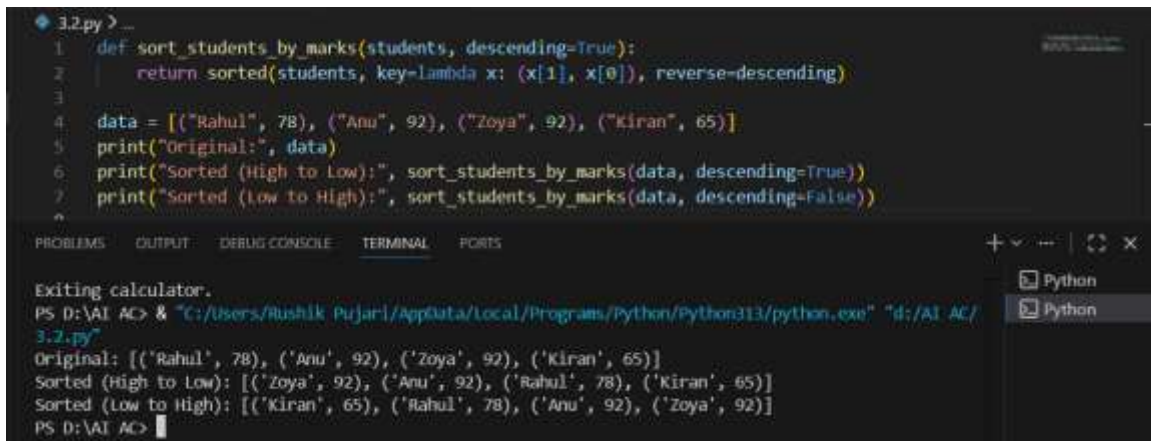
Task Description-2

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:



```
3.2.py > _
1 def sort_students_by_marks(students, descending=True):
2     return sorted(students, key=lambda x: (x[1], x[0]), reverse=descending)
3
4 data = [("Rahul", 78), ("Anu", 92), ("Zoya", 92), ("Kiran", 65)]
5 print("Original:", data)
6 print("Sorted (High to Low):", sort_students_by_marks(data, descending=True))
7 print("Sorted (Low to High):", sort_students_by_marks(data, descending=False))
8
9
10
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Exiting calculator.
PS D:\AI AC> & "C:/Users/Rushik Pujari/AppData/Local/Programs/Python/Python313/python.exe" "d:/AI AC/3.2.py"
Original: [('Rahul', 78), ('Anu', 92), ('Zoya', 92), ('Kiran', 65)]
Sorted (High to Low): [('Zoya', 92), ('Anu', 92), ('Rahul', 78), ('Kiran', 65)]
Sorted (Low to High): [('Kiran', 65), ('Rahul', 78), ('Anu', 92), ('Zoya', 92)]
PS D:\AI AC>
```

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:

```
3.2.py > ...
1  def is_prime(n):
2      if n <= 1:
3          return False
4      if n <= 3:
5          return True
6      if n % 2 == 0 or n % 3 == 0:
7          return False
8
9      i = 5
10     while i * i <= n:
11         if n % i == 0 or n % (i + 2) == 0:
12             return False
13         i += 6
14     return True
15
16 tests = [0, 1, 2, 3, 4, 5, 9, 11, 49, 97]
17 for t in tests:
18     print(t, "->", is_prime(t))
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
3.2.py
0 -> False
1 -> False
2 -> True
3 -> True
4 -> False
5 -> True
9 -> False
11 -> True
49 -> False
97 -> True
PS D:\AI AC>
```

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:

```
12.py > calculate_grade
1 def calculate_grade(percentage):
2     if percentage >= 90:
3         return "A+"
4     elif percentage >= 80:
5         return "A"
6     elif percentage >= 70:
7         return "B"
8     elif percentage >= 60:
9         return "C"
10    elif percentage >= 50:
11        return "D"
12    else:
13        return "F"
14 def student_grading_system():
15     print("=== Student Grading System ===")
16     try:
17         n = int(input("Enter number of subjects: "))
18         if n <= 0:
19             print("Number of subjects must be positive.")
20             return
21     except ValueError:
22         print("Invalid input. Enter an integer.")
23         return
24     marks = []
25     for i in range(1, n + 1):
26         try:
27             m = float(input(f"Enter marks for subject {i} (0-100): "))
28             if not (0 <= m <= 100):
29                 print("Marks must be between 0 and 100.")
30                 return
31             marks.append(m)
32         except ValueError:
33             print("Invalid marks input.")
34             return
35     total = sum(marks)
36     percentage = total / n
37     grade = calculate_grade(percentage)
38     print("\n--- Result ---")
39     print(f"Total Marks: {total:.2f} / {n*100}")
40     print(f"Percentage: {percentage:.2f}%")
41     print(f"Grade: {grade}")
42     student_grading_system()
43
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Enter number of subjects: 5
Invalid input. Enter an integer.
PS D:\AI AC> 5
PS D:\AI AC> "C:\Users\Rushik Pujari\AppData\Local\Programs\Python\Python313\python.exe" "d:/AI AC/3.2.py"
=== Student Grading System ===
Enter number of subjects: 5
Enter marks for subject 1 (0-100): 79
Enter marks for subject 2 (0-100): 57
Enter marks for subject 3 (0-100): 84
Enter marks for subject 4 (0-100): 97
Enter marks for subject 5 (0-100): 64

--- Result ---
Total Marks: 381.00 / 500
Percentage: 76.20%
Grade: B
```

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:

```
32.py 7 1 use_converter
1 def km_to_miles(km):
2     return km * 0.621371
3 def miles_to_km(miles):
4     return miles * 1.609344
5
6 def unit_converter():
7     print("--- Unit Converter (Km <-> Miles) ---")
8     print("1) Kilometers to Miles")
9     print("2) Miles to Kilometers")
10    print("3) Exit")
11    while True:
12        choice = input("\nChoose an option (1/2/3): ").strip()
13        if choice == '3':
14            print("Exiting converter.")
15            break
16        if choice not in ['1', '2']:
17            print("Invalid option, try again.")
18            continue
19        try:
20            value = float(input("Enter value: "))
21            if value < 0:
22                print("Distance cannot be negative.")
23                continue
24        except ValueError:
25            print("Invalid numeric input.")
26            continue
27        if choice == '1':
28            print(f"{value} km = {km_to_miles(value):.6f} miles")
29        else:
30            print(f"{value} miles = {miles_to_km(value):.6f} km")
31    unit_converter()
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

--- Unit Converter (Km <-> Miles) ---
1) Kilometers to Miles
1) Kilometers to Miles
2) Miles to Kilometers
3) Exit

Choose an option (1/2/3): 2
Enter value: 15
15.0 miles = 24.140160 km

Choose an option (1/2/3): 1
Enter value: 67
67.0 km = 41.631857 miles

Choose an option (1/2/3): 3
Exiting converter.
PS D:\AI AC>