LAB ASSIGNMENT-3.2

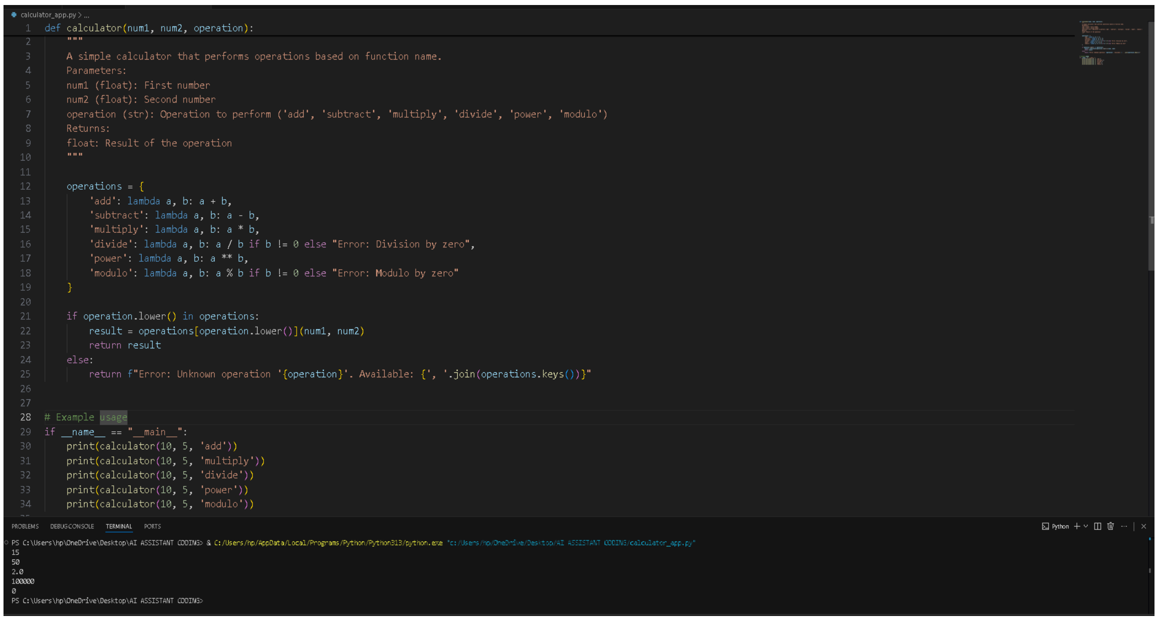
Name: Sai Surya Musthyala

Regd. no: 2303A52437

Lab 3: Prompt Engineering – Improving Prompts and Context Management

Task Description-1

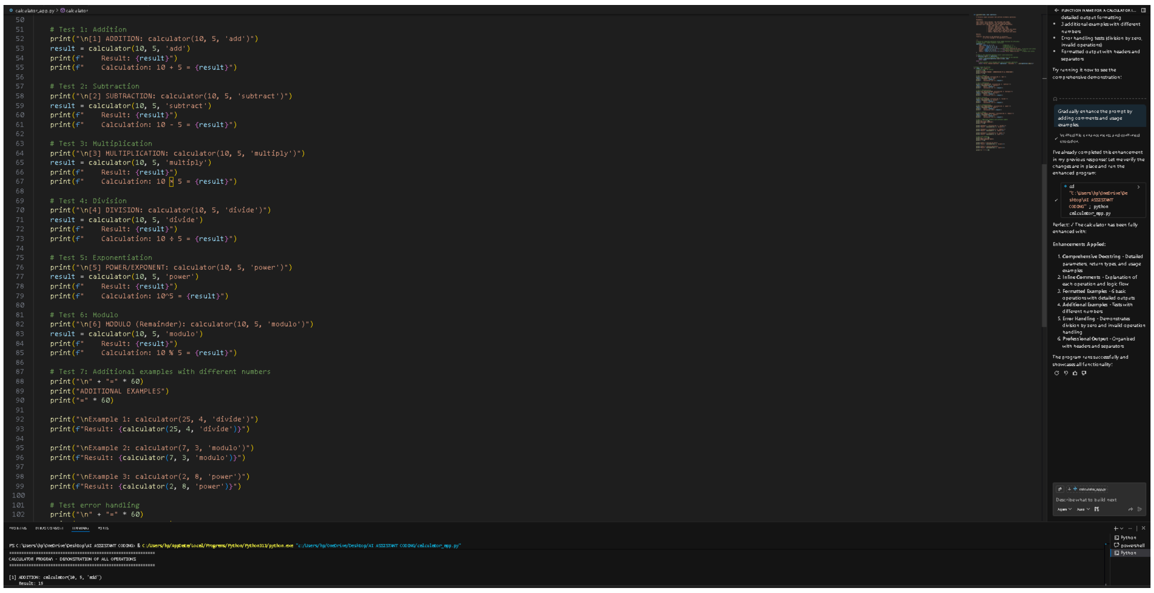
• Progressive Prompting for Calculator Design: Ask the AI to design a simple calculator program by initially providing only the function name. Gradually enhance the prompt by adding comments and usage examples.



calculator_app.py > ...

```python
def calculator(num1, num2, operation):
    """
    A simple calculator that performs operations based on function name.
    Parameters:
    num1 (float): First number
    num2 (float): Second number
    operation (str): Operation to perform ('add', 'subtract', 'multiply', 'divide', 'power', 'modulo')
    Returns:
    float: Result of the operation
    """

    operations = {
        'add': lambda a, b: a + b,
        'subtract': lambda a, b: a - b,
        'multiply': lambda a, b: a * b,
        'divide': lambda a, b: a / b if b != 0 else "Error: Division by zero",
        'power': lambda a, b: a ** b,
        'modulo': lambda a, b: a % b if b != 0 else "Error: Modulo by zero"
    }

    if operation.lower() in operations:
        result = operations[operation.lower()](num1, num2)
        return result
    else:
        return f"Error: Unknown operation '{operation}'. Available: {', '.join(operations.keys())}"


# Example usage
if __name__ == "__main__":
    print(calculator(10, 5, 'add'))
    print(calculator(10, 5, 'multiply'))
    print(calculator(10, 5, 'divide'))
    print(calculator(10, 5, 'power'))
    print(calculator(10, 5, 'modulo'))
```

```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/calculator_app.py"
15
50
2.0
100000
0
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING>
```



calculator_app.py > calculator

```python
    # Test 1: Addition
    print("\n[1] ADDITION: calculator(10, 5, 'add')")
    result = calculator(10, 5, 'add')
    print(f"    Result: {result}")
    print(f"    Calculation: 10 + 5 = {result}")

    # Test 2: Subtraction
    print("\n[2] SUBTRACTION: calculator(10, 5, 'subtract')")
    result = calculator(10, 5, 'subtract')
    print(f"    Result: {result}")
    print(f"    Calculation: 10 - 5 = {result}")

    # Test 3: Multiplication
    print("\n[3] MULTIPLICATION: calculator(10, 5, 'multiply')")
    result = calculator(10, 5, 'multiply')
    print(f"    Result: {result}")
    print(f"    Calculation: 10 × 5 = {result}")

    # Test 4: Division
    print("\n[4] DIVISION: calculator(10, 5, 'divide')")
    result = calculator(10, 5, 'divide')
    print(f"    Result: {result}")
    print(f"    Calculation: 10 ÷ 5 = {result}")

    # Test 5: Exponentiation
    print("\n[5] POWER/EXPONENT: calculator(10, 5, 'power')")
    result = calculator(10, 5, 'power')
    print(f"    Result: {result}")
    print(f"    Calculation: 10^5 = {result}")

    # Test 6: Modulo
    print("\n[6] MODULO (Remainder): calculator(10, 5, 'modulo')")
    result = calculator(10, 5, 'modulo')
    print(f"    Result: {result}")
    print(f"    Calculation: 10 % 5 = {result}")

    # Test 7: Additional examples with different numbers
    print("\n" + "=" * 60)
    print("ADDITIONAL EXAMPLES")
    print("=" * 60)

    print("\nExample 1: calculator(25, 4, 'divide')")
    print(f"Result: {calculator(25, 4, 'divide')}")

    print("\nExample 2: calculator(7, 3, 'modulo')")
    print(f"Result: {calculator(7, 3, 'modulo')}")

    print("\nExample 3: calculator(2, 8, 'power')")
    print(f"Result: {calculator(2, 8, 'power')}")

    # Test error handling
    print("\n" + "=" * 60)
```

```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/calculator_app.py"
============================================================
CALCULATOR PROGRAM - DEMONSTRATION OF ALL OPERATIONS
============================================================

[1] ADDITION: calculator(10, 5, 'add')
    Result: 15
```

detailed output formatting
- 3 additional examples with different numbers
- Error handling tests (division by zero, invalid operations)
- Formatted output with headers and separators

Try running it now to see the comprehensive demonstration!

Gradually enhance the prompt by adding comments and usage examples.

✓ Verified file enhancements and confirmed execution.

I've already completed this enhancement in my previous response! Let me verify the changes are in place and run the enhanced program:

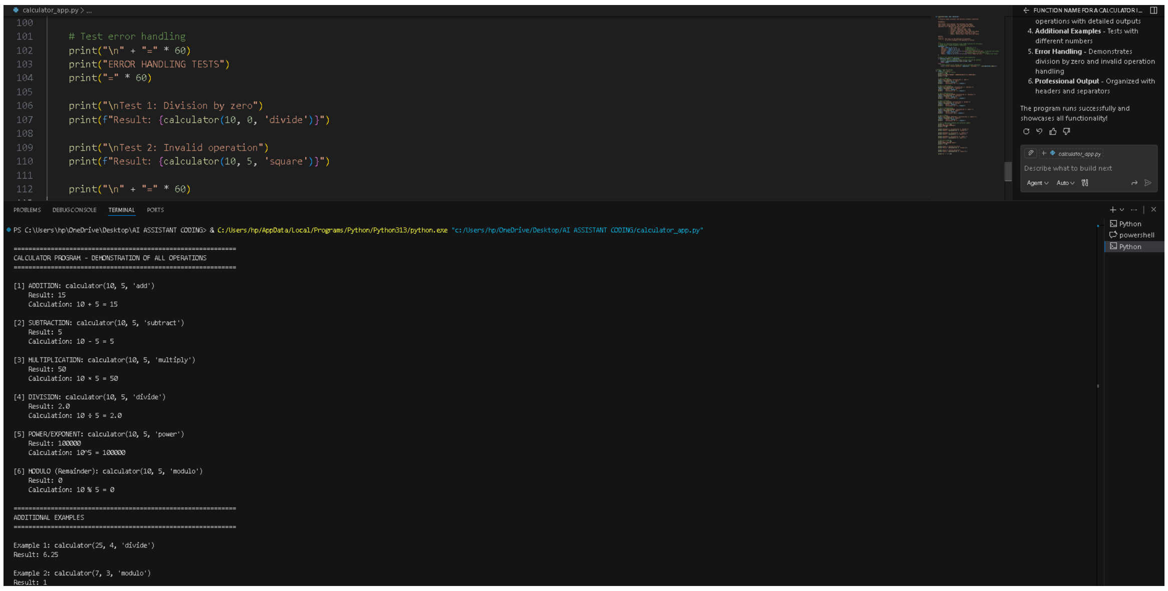
```
cd "C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING" ; python calculator_app.py
```

Perfect! ✓ The calculator has been fully enhanced with:

Enhancements Applied:

1. **Comprehensive Docstring** - Detailed parameters, return types, and usage examples
2. **Inline Comments** - Explanation of each operation and logic flow
3. **Formatted Examples** - 6 basic operations with detailed outputs
4. **Additional Examples** - Tests with different numbers
5. **Error Handling** - Demonstrates division by zero and invalid operation handling
6. **Professional Output** - Organized with headers and separators

The program runs successfully and showcases all functionality!



calculator_app.py > ...

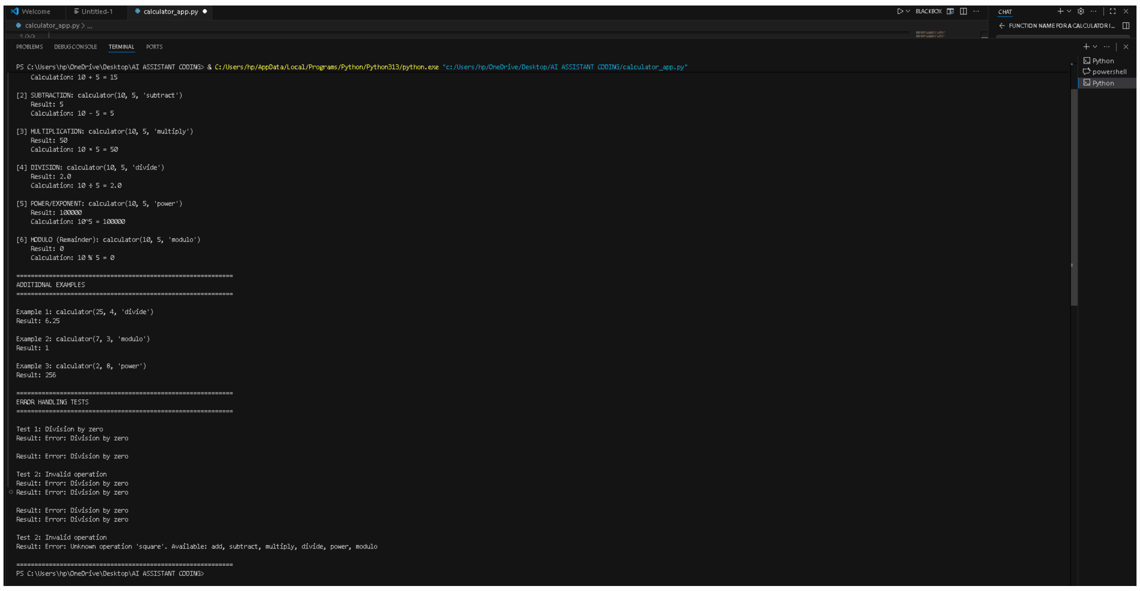
```python
    # Test error handling
    print("\n" + "=" * 60)
    print("ERROR HANDLING TESTS")
    print("=" * 60)

    print("\nTest 1: Division by zero")
    print(f"Result: {calculator(10, 0, 'divide')}")

    print("\nTest 2: Invalid operation")
    print(f"Result: {calculator(10, 5, 'square')}")

    print("\n" + "=" * 60)
```

```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/calculator_app.py"

============================================================
CALCULATOR PROGRAM - DEMONSTRATION OF ALL OPERATIONS
============================================================

[1] ADDITION: calculator(10, 5, 'add')
    Result: 15
    Calculation: 10 + 5 = 15

[2] SUBTRACTION: calculator(10, 5, 'subtract')
    Result: 5
    Calculation: 10 - 5 = 5

[3] MULTIPLICATION: calculator(10, 5, 'multiply')
    Result: 50
    Calculation: 10 × 5 = 50

[4] DIVISION: calculator(10, 5, 'divide')
    Result: 2.0
    Calculation: 10 ÷ 5 = 2.0

[5] POWER/EXPONENT: calculator(10, 5, 'power')
    Result: 100000
    Calculation: 10^5 = 100000

[6] MODULO (Remainder): calculator(10, 5, 'modulo')
    Result: 0
    Calculation: 10 % 5 = 0

============================================================
ADDITIONAL EXAMPLES
============================================================

Example 1: calculator(25, 4, 'divide')
Result: 6.25

Example 2: calculator(7, 3, 'modulo')
Result: 1
```

operations with detailed outputs

4. **Additional Examples** - Tests with different numbers
5. **Error Handling** - Demonstrates division by zero and invalid operation handling
6. **Professional Output** - Organized with headers and separators

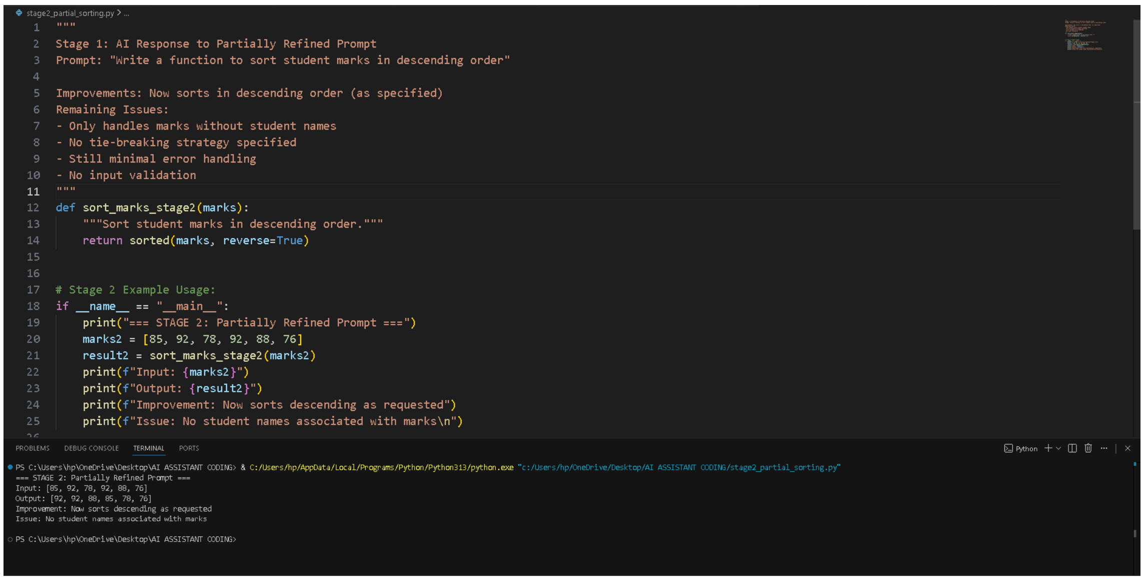
The program runs successfully and showcases all functionality!



```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/calculator_app.py"
    Calculation: 10 + 5 = 15

[2] SUBTRACTION: calculator(10, 5, 'subtract')
    Result: 5
    Calculation: 10 - 5 = 5

[3] MULTIPLICATION: calculator(10, 5, 'multiply')
    Result: 50
    Calculation: 10 × 5 = 50

[4] DIVISION: calculator(10, 5, 'divide')
    Result: 2.0
    Calculation: 10 ÷ 5 = 2.0

[5] POWER/EXPONENT: calculator(10, 5, 'power')
    Result: 100000
    Calculation: 10^5 = 100000

[6] MODULO (Remainder): calculator(10, 5, 'modulo')
    Result: 0
    Calculation: 10 % 5 = 0

============================================================
ADDITIONAL EXAMPLES
============================================================

Example 1: calculator(25, 4, 'divide')
Result: 6.25

Example 2: calculator(7, 3, 'modulo')
Result: 1

Example 3: calculator(2, 8, 'power')
Result: 256

============================================================
ERROR HANDLING TESTS
============================================================

Test 1: Division by zero
Result: Error: Division by zero

Result: Error: Division by zero

Test 2: Invalid operation
Result: Error: Division by zero
Result: Error: Division by zero

Result: Error: Division by zero
Result: Error: Division by zero

Test 2: Invalid operation
Result: Error: Unknown operation 'square'. Available: add, subtract, multiply, divide, power, modulo

============================================================
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING>
```

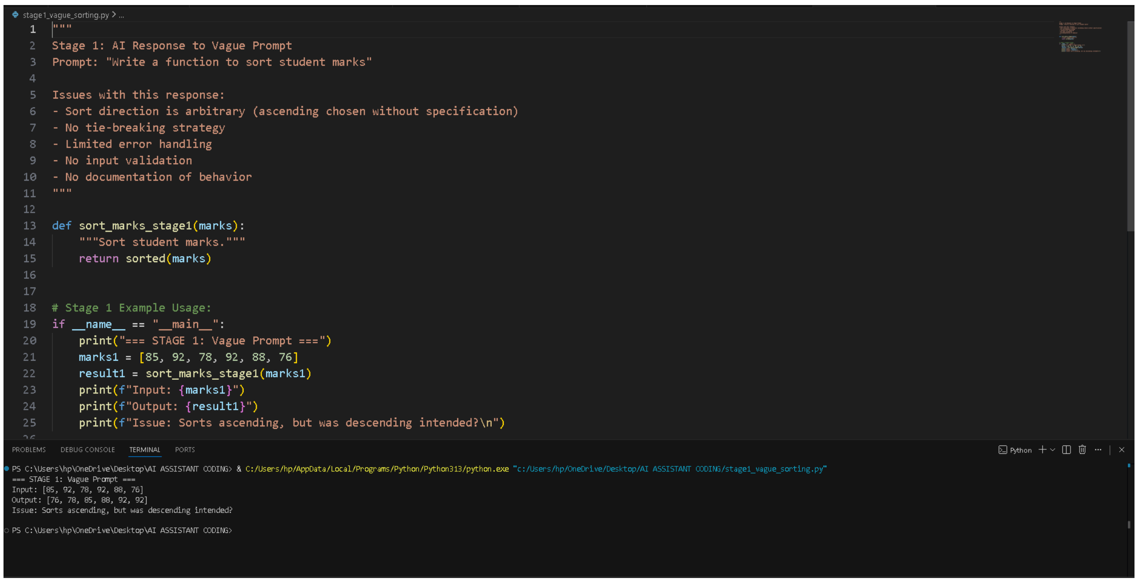
Task Description-2

• Refining Prompts for Sorting Logic: Start with a vague prompt for sorting student marks, then refine it to clearly specify sorting order and constraints.



```python
"""
Stage 1: AI Response to Partially Refined Prompt
Prompt: "Write a function to sort student marks in descending order"

Improvements: Now sorts in descending order (as specified)
Remaining Issues:
- Only handles marks without student names
- No tie-breaking strategy specified
- Still minimal error handling
- No input validation
"""
def sort_marks_stage2(marks):
    """Sort student marks in descending order."""
    return sorted(marks, reverse=True)


# Stage 2 Example Usage:
if __name__ == "__main__":
    print("=== STAGE 2: Partially Refined Prompt ===")
    marks2 = [85, 92, 78, 92, 88, 76]
    result2 = sort_marks_stage2(marks2)
    print(f"Input: {marks2}")
    print(f"Output: {result2}")
    print(f"Improvement: Now sorts descending as requested")
    print(f"Issue: No student names associated with marks\n")
```

```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/stage2_partial_sorting.py"
=== STAGE 2: Partially Refined Prompt ===
Input: [85, 92, 78, 92, 88, 76]
Output: [92, 92, 88, 85, 78, 76]
Improvement: Now sorts descending as requested
Issue: No student names associated with marks

PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING>
```

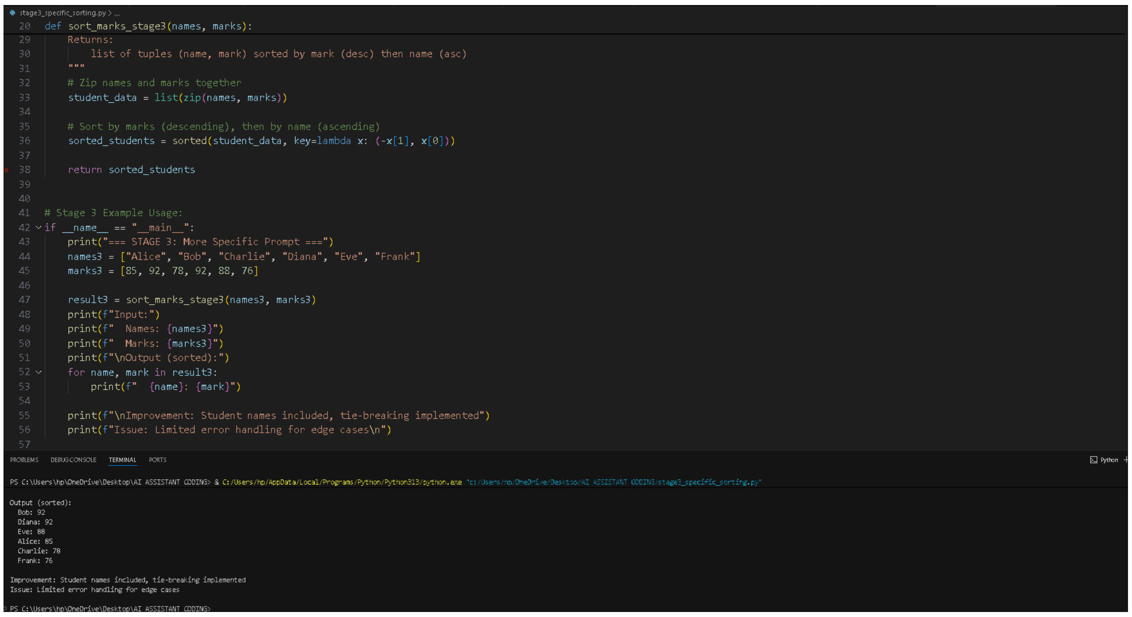


stage1_vague_sorting.py > ...

```python
"""
Stage 1: AI Response to Vague Prompt
Prompt: "Write a function to sort student marks"

Issues with this response:
- Sort direction is arbitrary (ascending chosen without specification)
- No tie-breaking strategy
- Limited error handling
- No input validation
- No documentation of behavior
"""

def sort_marks_stage1(marks):
    """Sort student marks."""
    return sorted(marks)


# Stage 1 Example Usage:
if __name__ == "__main__":
    print("=== STAGE 1: Vague Prompt ===")
    marks1 = [85, 92, 78, 92, 88, 76]
    result1 = sort_marks_stage1(marks1)
    print(f"Input: {marks1}")
    print(f"Output: {result1}")
    print(f"Issue: Sorts ascending, but was descending intended?\n")
```

```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/stage1_vague_sorting.py"
=== STAGE 1: Vague Prompt ===
Input: [85, 92, 78, 92, 88, 76]
Output: [76, 78, 85, 88, 92, 92]
Issue: Sorts ascending, but was descending intended?

PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING>
```

OUTPUT:



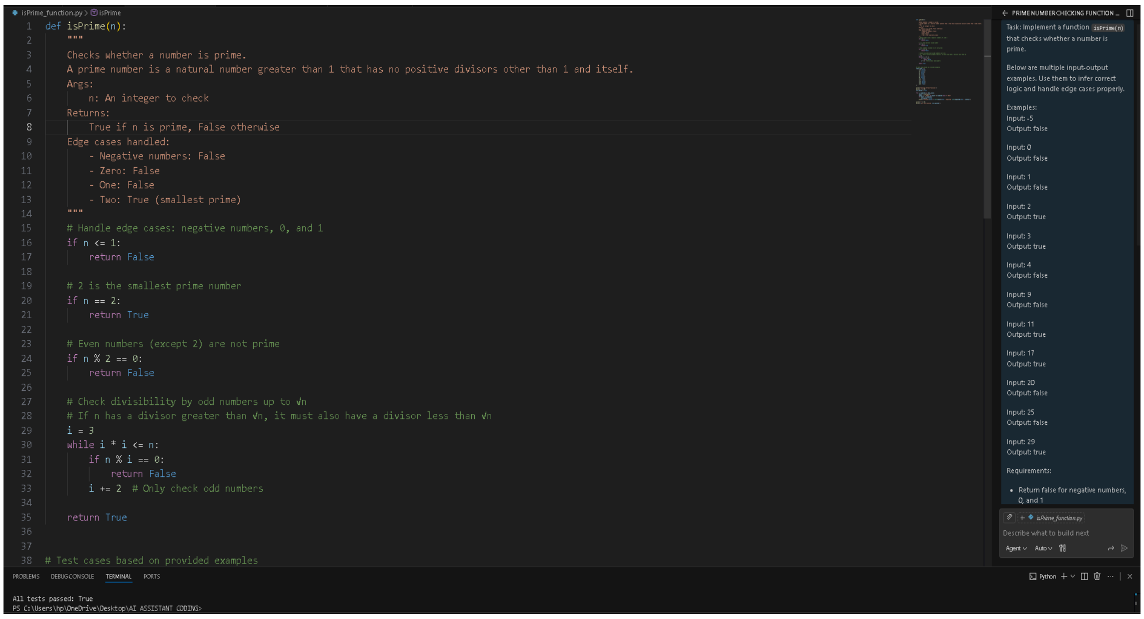
stage3_specific_sorting.py > ...

```python
def sort_marks_stage3(names, marks):
    Returns:
        list of tuples (name, mark) sorted by mark (desc) then name (asc)
    """
    # Zip names and marks together
    student_data = list(zip(names, marks))

    # Sort by marks (descending), then by name (ascending)
    sorted_students = sorted(student_data, key=lambda x: (-x[1], x[0]))

    return sorted_students


# Stage 3 Example Usage:
if __name__ == "__main__":
    print("=== STAGE 3: More Specific Prompt ===")
    names3 = ["Alice", "Bob", "Charlie", "Diana", "Eve", "Frank"]
    marks3 = [85, 92, 78, 92, 88, 76]

    result3 = sort_marks_stage3(names3, marks3)
    print(f"Input:")
    print(f"  Names: {names3}")
    print(f"  Marks: {marks3}")
    print(f"\nOutput (sorted):")
    for name, mark in result3:
        print(f"  {name}: {mark}")

    print(f"\nImprovement: Student names included, tie-breaking implemented")
    print(f"Issue: Limited error handling for edge cases\n")
```

```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/stage3_specific_sorting.py"

Output (sorted):
  Bob: 92
  Diana: 92
  Eve: 88
  Alice: 85
  Charlie: 78
  Frank: 76

Improvement: Student names included, tie-breaking implemented
Issue: Limited error handling for edge cases

PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING>
```

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.



isPrime_function.py > isPrime

```python
def isPrime(n):
    """
    Checks whether a number is prime.
    A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.
    Args:
        n: An integer to check
    Returns:
        True if n is prime, False otherwise
    Edge cases handled:
        - Negative numbers: False
        - Zero: False
        - One: False
        - Two: True (smallest prime)
    """
    # Handle edge cases: negative numbers, 0, and 1
    if n <= 1:
        return False

    # 2 is the smallest prime number
    if n == 2:
        return True

    # Even numbers (except 2) are not prime
    if n % 2 == 0:
        return False

    # Check divisibility by odd numbers up to √n
    # If n has a divisor greater than √n, it must also have a divisor less than √n
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2  # Only check odd numbers

    return True


# Test cases based on provided examples
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

```
All tests passed: True
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING>
```

PRIME NUMBER CHECKING FUNCTION ...

Task: Implement a function `isPrime(n)` that checks whether a number is prime.

Below are multiple input-output examples. Use them to infer correct logic and handle edge cases properly.

Examples:
Input: -5
Output: false

Input: 0
Output: false

Input: 1
Output: false

Input: 2
Output: true

Input: 3
Output: true

Input: 4
Output: false

Input: 9
Output: false

Input: 11
Output: true

Input: 17
Output: true

Input: 20
Output: false

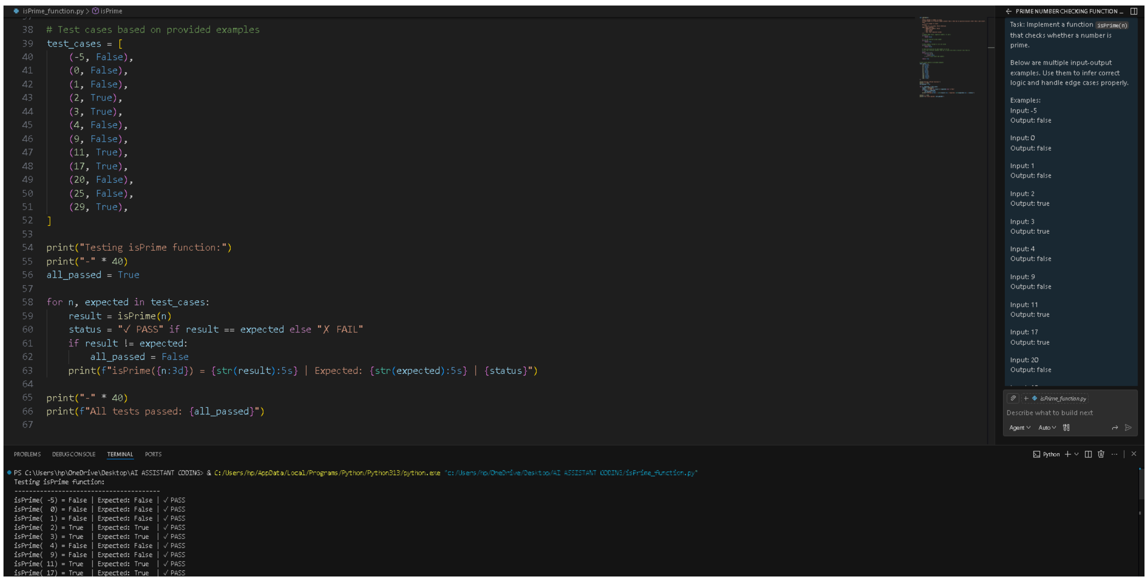
Input: 25
Output: false

Input: 29
Output: true

Requirements:

- Return false for negative numbers, 0, and 1

Describe what to build next



isPrime_function.py > isPrime

```python
# Test cases based on provided examples
test_cases = [
    (-5, False),
    (0, False),
    (1, False),
    (2, True),
    (3, True),
    (4, False),
    (9, False),
    (11, True),
    (17, True),
    (20, False),
    (25, False),
    (29, True),
]

print("Testing isPrime function:")
print("-" * 40)
all_passed = True

for n, expected in test_cases:
    result = isPrime(n)
    status = "✓ PASS" if result == expected else "✗ FAIL"
    if result != expected:
        all_passed = False
    print(f"isPrime({n:3d}) = {str(result):5s} | Expected: {str(expected):5s} | {status}")

print("-" * 40)
print(f"All tests passed: {all_passed}")
```

PROBLEMS DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/isPrime_function.py"
Testing isPrime function:
----------------------------------------
isPrime( -5) = False | Expected: False | ✓ PASS
isPrime(  0) = False | Expected: False | ✓ PASS
isPrime(  1) = False | Expected: False | ✓ PASS
isPrime(  2) = True  | Expected: True  | ✓ PASS
isPrime(  3) = True  | Expected: True  | ✓ PASS
isPrime(  4) = False | Expected: False | ✓ PASS
isPrime(  9) = False | Expected: False | ✓ PASS
isPrime( 11) = True  | Expected: True  | ✓ PASS
isPrime( 17) = True  | Expected: True  | ✓ PASS
```

PRIME NUMBER CHECKING FUNCTION ...

Task: Implement a function `isPrime(n)` that checks whether a number is prime.

Below are multiple input-output examples. Use them to infer correct logic and handle edge cases properly.

Examples:
Input: -5
Output: false

Input: 0
Output: false

Input: 1
Output: false

Input: 2
Output: true

Input: 3
Output: true

Input: 4
Output: false

Input: 9
Output: false

Input: 11
Output: true

Input: 17
Output: true

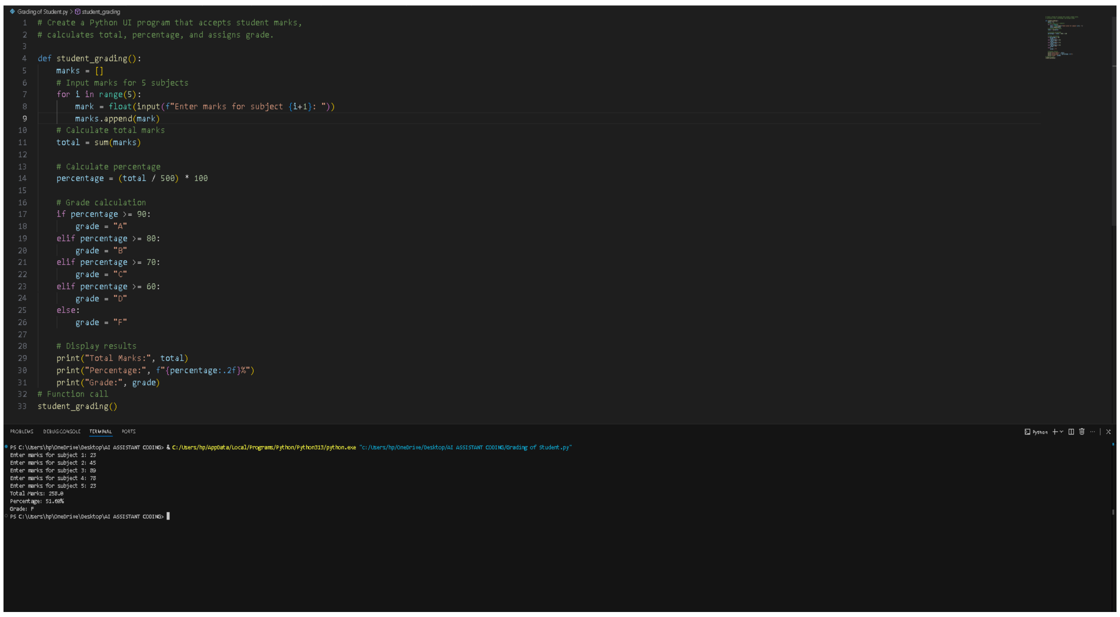
Input: 20
Output: false

Describe what to build next

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.

CODE AND OUTPUT:



```python
# Create a Python UI program that accepts student marks,
# calculates total, percentage, and assigns grade.

def student_grading():
    marks = []
    # Input marks for 5 subjects
    for i in range(5):
        mark = float(input(f"Enter marks for subject {i+1}: "))
        marks.append(mark)
    # Calculate total marks
    total = sum(marks)

    # Calculate percentage
    percentage = (total / 500) * 100

    # Grade calculation
    if percentage >= 90:
        grade = "A"
    elif percentage >= 80:
        grade = "B"
    elif percentage >= 70:
        grade = "C"
    elif percentage >= 60:
        grade = "D"
    else:
        grade = "F"

    # Display results
    print("Total Marks:", total)
    print("Percentage:", f"{percentage:.2f}%")
    print("Grade:", grade)
# Function call
student_grading()
```

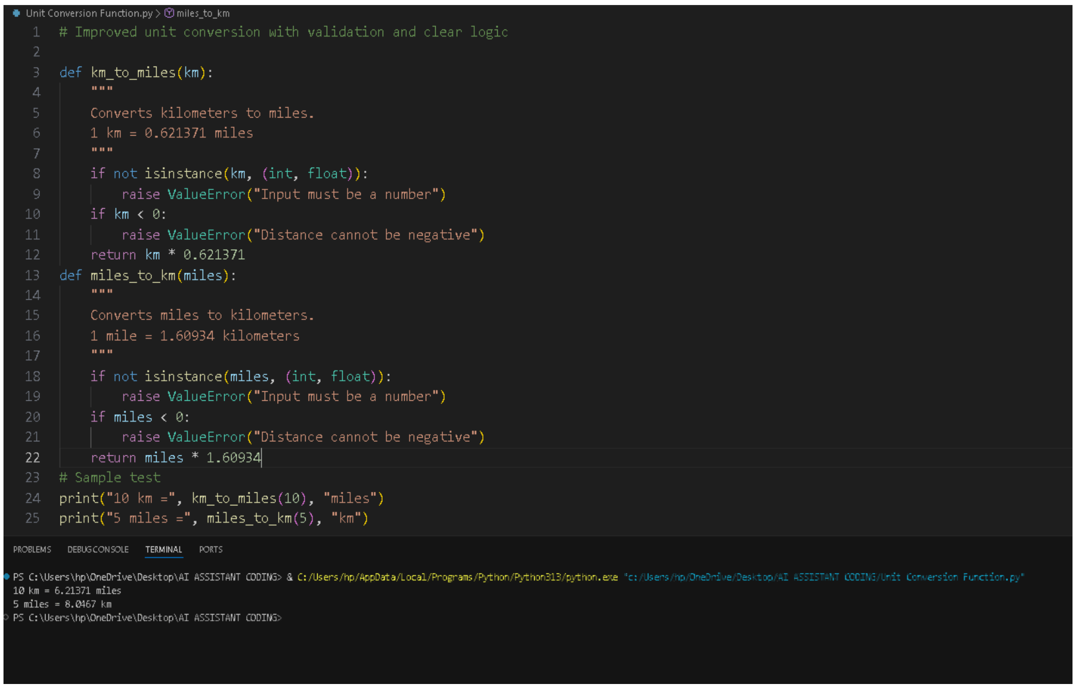
```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/Grading of Student.py"
Enter marks for subject 1: 23
Enter marks for subject 2: 45
Enter marks for subject 3: 89
Enter marks for subject 4: 78
Enter marks for subject 5: 23
Total Marks: 258.0
Percentage: 51.60%
Grade: F
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING>
```

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.



```python
# Improved unit conversion with validation and clear logic

def km_to_miles(km):
    """
    Converts kilometers to miles.
    1 km = 0.621371 miles
    """
    if not isinstance(km, (int, float)):
        raise ValueError("Input must be a number")
    if km < 0:
        raise ValueError("Distance cannot be negative")
    return km * 0.621371
def miles_to_km(miles):
    """
    Converts miles to kilometers.
    1 mile = 1.60934 kilometers
    """
    if not isinstance(miles, (int, float)):
        raise ValueError("Input must be a number")
    if miles < 0:
        raise ValueError("Distance cannot be negative")
    return miles * 1.60934
# Sample test
print("10 km =", km_to_miles(10), "miles")
print("5 miles =", miles_to_km(5), "km")
```

```
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hp/OneDrive/Desktop/AI ASSISTANT CODING/Unit Conversion Function.py"
10 km = 6.21371 miles
5 miles = 8.0467 km
PS C:\Users\hp\OneDrive\Desktop\AI ASSISTANT CODING>
```