

Name: Alluri Chaithra ID:2403A51105 Batch:06

Task Description#1 Basic Docstring Generation

- Write python function to return sum of even and odd numbers in the given list.
- Incorporate manual **docstring** in code with Google Style
- Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function.
- Compare the AI-generated docstring with your manually written one.

• Prompt:

Generate docstring for sum_even_odd function

"Generate a Google-style docstring for a Python function sum_even_odd(numbers) that takes a list of integers and returns a tuple containing the sum of even numbers and the sum of odd numbers." give the code and output

```
def sum_even_odd(numbers):  
    """Calculate the sum of even and odd numbers from a list of integers.  
  
    This function takes a list of integers and returns a tuple containing  
    the sum of all even numbers and the sum of all odd numbers.  
  
    Args:  
        numbers (list[int]): A list of integers to be processed.  
  
    Returns:  
        tuple[int, int]: A tuple containing (sum_of_even_numbers, sum_of_odd_numbers).  
        If the input list is empty, returns (0, 0).  
  
    Raises:  
        TypeError: If the input is not a list or contains non-integer values.  
  
    Examples:  
        >>> sum_even_odd([1, 2, 3, 4, 5])  
        (6, 9)  
        >>> sum_even_odd([2, 4, 6])  
        (12, 0)  
        >>> sum_even_odd([1, 3, 5])  
        (0, 9)  
        >>> sum_even_odd([])  
        (0, 0)  
    """  
    if not isinstance(numbers, list):  
        raise TypeError("Input must be a list")  
  
    even_sum = 0
```

Code:

```
    odd_sum = 0  
  
    for num in numbers:  
        if not isinstance(num, int):  
            raise TypeError("All elements in the list must be integers")  
  
        if num % 2 == 0:  
            even_sum += num  
        else:  
            odd_sum += num  
  
    return (even_sum, odd_sum)  
  
# Test the function with various inputs  
if __name__ == "__main__":  
    # Test cases  
    test_cases = [  
        [1, 2, 3, 4, 5],  
        [2, 4, 6, 8],  
        [1, 3, 5, 7],  
        [10, 15, 20, 25, 30],  
        [],  
        [0, -1, -2, -3, -4]  
    ]  
  
    print("Testing sum_even_odd function:")  
    print("=" * 40)  
  
    for i, test_list in enumerate(test_cases, 1):  
        result = sum_even_odd(test_list)  
        print(f"Test {i}: {test_list}")  
        print(f"Result: Even sum = {result[0]}, Odd sum = {result[1]}")  
        print(f"Tuple: {result}")
```

Expected Outcome#1: Students understand how AI can produce function-level documentation.

```
print(f"Tuple: {result}")
print("-" * 30)
```

Output:

≡ text

```
Testing sum_even_odd function:
=====
Test 1: [1, 2, 3, 4, 5]
Result: Even sum = 6, Odd sum = 9
Tuple: (6, 9)
-----
Test 2: [2, 4, 6, 8]
Result: Even sum = 20, Odd sum = 0
Tuple: (20, 0)
-----
Test 3: [1, 3, 5, 7]
Result: Even sum = 0, Odd sum = 16
Tuple: (0, 16)
-----
Test 4: [10, 15, 20, 25, 30]
Result: Even sum = 60, Odd sum = 40
Tuple: (60, 40)
-----
Test 5: []
Result: Even sum = 0, Odd sum = 0
Tuple: (0, 0)
-----
Test 6: [0, -1, -2, -3, -4]
Result: Even sum = -6, Odd sum = -4
Tuple: (-6, -4)
=====
```

Output:

Task Description#2 Automatic Inline Comments

- Write python program for **sru_student** class with attributes like name, roll no., hostel_status and **fee_update** method and **display_details** method.
- Write comments manually for each line/code block
- Ask an AI tool to add inline comments explaining each line/step.
- Compare the AI-generated comments with your manually written one.

• Prompt:

Add detailed inline comments to this Python program explaining each line and step, especially focusing on class, attributes, methods (fee_update, display_details), and object usage for beginners. give the code and output

```

class Student:
    """Student class with fee management."""

    total_students = 0 # Class variable

    def __init__(self, name, student_id, course, initial_fee=0):
        """Constructor - creates new student object."""
        self.name = name # Instance attribute
        self.student_id = student_id
        self.course = course
        self.fee_balance = initial_fee
        Student.total_students += 1
        print(f"Student '{self.name}' enrolled!")

    def fee_update(self, amount, operation="add"):
        """Update student's fee balance."""
        if operation == "add":
            self.fee_balance += amount
            print(f"Added ${amount}. New balance: ${self.fee_balance}")
        elif operation == "subtract":
            self.fee_balance -= amount
            print(f"Subtracted ${amount}. New balance: ${self.fee_balance}")
        else:
            print("Invalid operation!")
        return self.fee_balance

    def display_details(self):
        """Display student information."""
        print(f"\nName: {self.name}")
        print(f"ID: {self.student_id}")
        print(f"Course: {self.course}")
        print(f"Balance: ${self.fee_balance}")

```

Code:

```

@classmethod
def get_total_students(cls):
    """Get total number of students."""
    return cls.total_students

# Main program
if __name__ == "__main__":
    # Create student objects
    student1 = Student("Alice", "STU001", "CS", 1000)
    student2 = Student("Bob", "STU002", "Math", 500)
    student3 = Student("Carol", "STU003", "Physics")

    # Display details
    student1.display_details()
    student2.display_details()
    student3.display_details()

    # Fee operations
    student1.fee_update(200, "add")
    student2.fee_update(100, "subtract")
    student3.fee_update(300, "add")

    # Show updated details
    student1.display_details()
    student2.display_details()
    student3.display_details()

    # Class method
    print(f"Total students: {Student.get_total_students()}")

```

Expected Output#2: Students critically analyze AI-generated code comments.

```
Student 'Alice' enrolled!
Student 'Bob' enrolled!
Student 'Carol' enrolled!

Name: Alice
ID: STU001
Course: CS
Balance: $1000

Name: Bob
ID: STU002
Course: Math
Balance: $500

Name: Carol
ID: STU003
Course: Physics
Balance: $0

Added $200. New balance: $1200
Subtracted $100. New balance: $400
Added $300. New balance: $300

Name: Alice
ID: STU001
Course: CS
Balance: $1200

Name: Bob
ID: STU002
Course: Math
Balance: $400

Name: Carol
ID: STU003
Course: Physics
Balance: $300
```

Output:

Name: Bob
ID: STU002
Course: Math
Balance: \$400

Name: Carol
ID: STU003
Course: Physics
Balance: \$300

Total students: 3

Task Description#3

Subtract second number from first.

Parameters

a : float or int
 Number to subtract from.
b : float or int
 Number to subtract.

Returns

float or int
 Result of a - b.
"""

return a - b

def multiply(a, b):

"""

Multiply two numbers.

Parameters

a : float or int
 First number.
b : float or int
 Second number.

Returns

float or int
 Product of a and b.
"""

return a * b

```

def divide(a, b):
    """
    Divide first number by second.

    Parameters
    -----
    a : float or int
        Dividend.
    b : float or int
        Divisor.

    Returns
    -----
    float
        Quotient of a / b.

    Raises
    -----
    ValueError
        If b is zero.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero!")
    return a / b

def calculator_menu():
    """
    Interactive calculator menu.

    Returns
    -----
    None
    """
    print("=== Calculator ===")

```

- Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).
- Incorporate manual **docstring** in code with NumPy Style
- Use AI assistance to generate a module-level docstring + individual function docstrings.
- Compare the AI-generated docstring with your manually written one.

Prompt:

Generate a module-level docstring and NumPy-style docstrings for each function in this Python script that performs basic calculator operations (add, subtract, multiply, divide), give the code and output'

Expected Output#3: Students learn structured documentation for multi-function scripts

Code:

```
Subtract second number from first.
```

```
Parameters
```

```
-----
```

```
a : float or int  
    Number to subtract from.
```

```
b : float or int  
    Number to subtract.
```

```
Returns
```

```
-----
```

```
float or int  
    Result of a - b.
```

```
"""
```

```
return a - b
```

```
def multiply(a, b):
```

```
    """
```

```
    Multiply two numbers.
```

```
Parameters
```

```
-----
```

```
a : float or int  
    First number.
```

```
b : float or int  
    Second number.
```

```
Returns
```

```
-----
```

```
float or int  
    Product of a and b.
```

```
"""
```

```
return a * b
```



```

def divide(a, b):
    """
    Divide first number by second.

    Parameters
    -----
    a : float or int
        Dividend.
    b : float or int
        Divisor.

    Returns
    -----
    float
        Quotient of a / b.

    Raises
    -----
    ValueError
        If b is zero.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero!")
    return a / b


def calculator_menu():
    """
    Interactive calculator menu.

    Returns
    -----
    None
    """
    print("=== Calculator ===")

```

```

print("1. Add 2. Subtract 3. Multiply 4. Divide 5. Exit")

while True:
    choice = input("Choice (1-5): ")
    if choice == '5':
        break

    if choice in ['1', '2', '3', '4']:
        try:
            a = float(input("First number: "))
            b = float(input("Second number: "))

            if choice == '1':
                print(f"Result: {add(a, b)}")
            elif choice == '2':
                print(f"Result: {subtract(a, b)}")
            elif choice == '3':
                print(f"Result: {multiply(a, b)}")
            elif choice == '4':
                print(f"Result: {divide(a, b)}")
        except ValueError as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    # Demo
    print("Calculator Demo:")
    print(f"5 + 3 = {add(5, 3)}")
    print(f"10 - 4 = {subtract(10, 4)}")
    print(f"6 * 7 = {multiply(6, 7)}")
    print(f"15 / 3 = {divide(15, 3)}")

    try:
        divide(5, 0)
    except ValueError as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    # Demo
    print("Calculator Demo:")
    print(f"5 + 3 = {add(5, 3)}")
    print(f"10 - 4 = {subtract(10, 4)}")
    print(f"6 * 7 = {multiply(6, 7)}")
    print(f"15 / 3 = {divide(15, 3)}")

    try:
        divide(5, 0)
    except ValueError as e:
        print(f"Error: {e}")

    calculator_menu()

```

Output:

≡ text

```
Calculator Demo:
5 + 3 = 8
10 - 4 = 6
6 * 7 = 42
15 / 3 = 5.0
Error: Cannot divide by zero!
=== Calculator ===
1. Add 2. Subtract 3. Multiply 4. Divide 5. Exit
Choice (1-5): 1
First number: 10
Second number: 5
Result: 15.0
Choice (1-5): 4
First number: 20
Second number: 4
Result: 5.0
Choice (1-5): 5
```

Output:

Push documentation whole workspace as .md file in GitHub Repository

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots