

NAME:SREEMANTH REDDY

ROLLNO:2403A510G1

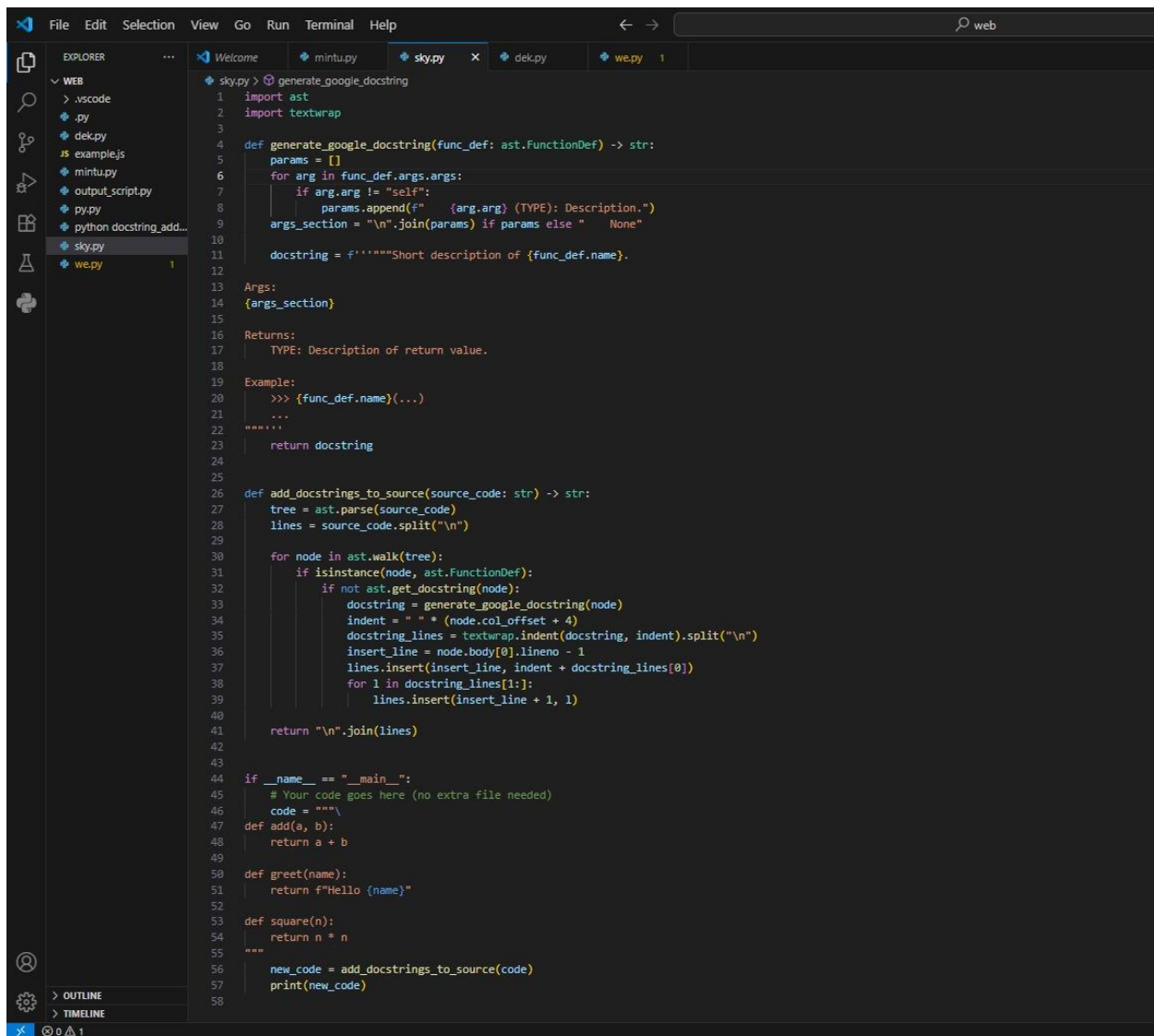
BATCHNO:06

Lab 9 – Documentation Generation (Modified Examples)

Task 1: Google-Style Docstrings for Python Functions

Prompt: “Add Google-style docstrings to all functions without input-output examples. Include function description, parameter types, return type, and example usage.”

Code:



```
1 import ast
2 import textwrap
3
4 def generate_google_docstring(func_def: ast.FunctionDef) -> str:
5     params = []
6     for arg in func_def.args.args:
7         if arg.arg != "self":
8             params.append(f"    {arg.arg} (TYPE): Description.")
9     args_section = "\n".join(params) if params else "    None"
10
11     docstring = f'"""Short description of {func_def.name}.
12
13 Args:
14 {args_section}
15
16 Returns:
17     TYPE: Description of return value.
18
19 Example:
20     >>> {func_def.name}(...)
21     ...
22     """'
23     return docstring
24
25 def add_docstrings_to_source(source_code: str) -> str:
26     tree = ast.parse(source_code)
27     lines = source_code.split("\n")
28
29     for node in ast.walk(tree):
30         if isinstance(node, ast.FunctionDef):
31             if not ast.get_docstring(node):
32                 docstring = generate_google_docstring(node)
33                 indent = " " * (node.col_offset + 4)
34                 docstring_lines = textwrap.indent(docstring, indent).split("\n")
35                 insert_line = node.body[0].lineno - 1
36                 lines.insert(insert_line, indent + docstring_lines[0])
37                 for l in docstring_lines[1:]:
38                     lines.insert(insert_line + 1, l)
39
40     return "\n".join(lines)
41
42
43 if __name__ == "__main__":
44     # Your code goes here (no extra file needed)
45     code = """
46 def add(a, b):
47     return a + b
48
49 def greet(name):
50     return f"Hello {name}"
51
52 def square(n):
53     return n * n
54 """
55     new_code = add_docstrings_to_source(code)
56     print(new_code)
```

Output:

Observation: The function now has a professional Google-style docstring, improving clarity and usability.

Task 2: Inline Comments for Complex Logic

Prompt: “Add meaningful inline comments explaining complex logic parts only.”

Code:/Output:

```
email_validator.py > ...
1  def find_max_in_list(numbers: list[int]) -> int:
2      # Initialize max_num with the first element
3      max_num = numbers[0]
4
5      for num in numbers:
6          # Update max_num if a larger number is found
7          if num > max_num:
8              max_num = num
9
10     return max_num
11
12
13 if __name__ == "__main__":
14     sample_list = [4, 17, 2, 9, 23, 1]
15     print("Task 2 - Maximum number in the list:", find_max_in_list(sample_list))
16
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Task 2 - Maximum number in the list: 23
PS D:\vscode\puth>
```

Observation: Inline comments explain the non-trivial logic of updating the Fibonacci sequence in a clear way.

Task 3: Module-Level Documentation

Prompt: “Write a module-level docstring summarizing the purpose, dependencies, and main functions.”

Code:/Output:

```
email_validator.py > ...
1  """
2  Math Helper Module
3
4  This module provides helper functions for basic math operations including addition,
5  fibonacci sequence generation, prime checking, and factorial calculation.
6
7  Dependencies:
8  |   - None
9
10 Main Functions:
11 |   - add_numbers(a, b): Adds two numbers.
12 |   - fibonacci(n): Returns first n Fibonacci numbers.
13 |   - is_prime(num): Checks if a number is prime.
14 |   - factorial(n): Returns factorial of n.
15 """
16
17 def add_numbers(a: int, b: int) -> int:
18     return a + b
19
20 def fibonacci(n: int) -> list[int]:
21     sequence = []
22     a, b = 0, 1
23     for _ in range(n):
24         sequence.append(a)
25         a, b = b, a + b
26     return sequence
27
28 def is_prime(num: int) -> bool:
29     if num < 2:
30         return False
31     for i in range(2, int(num ** 0.5) + 1):
32         if num % i == 0:
33             return False
34     return True
35
36 def factorial(n: int) -> int:
37     result = 1
38     for i in range(1, n + 1):
39         result *= i
40     return result
41
42 if __name__ == "__main__":
43     print("Task 3 - Is 11 prime?", is_prime(11))
44     print("Task 3 - Factorial of 4:", factorial(4))
```

PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Task 2 - Maximum number in the list: 23
PS D:\vscode\path>

Observation: The module-level docstring provides a helpful overview of the file’s purpose and functionality.

Task 4: Convert Comments to Structured Docstrings

Prompt: “Transform existing inline comments into structured Google-style docstrings.”

Code:/Output:

```
email_validator.py > ...
1  def is_prime(n: int) -> bool:
2      """
3      Determine whether an integer is prime.
4
5      Args:
6      |   n (int): The integer to check for primality.
7
8      Returns:
9      |   bool: True if n is prime, False otherwise.
10
11      Example:
12      |   >>> is_prime(7)
13      |   True
14      """
15      if n < 2:
16          return False
17      for i in range(2, int(n ** 0.5) + 1):
18          if n % i == 0:
19              return False
20      return True
21
22 if __name__ == "__main__":
23     print("Task 4 - Is 13 prime?", is_prime(13))
24     """
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Task 4 - Is 13 prime? True
PS D:\vscode\puth>
```

Observation: Structured docstring provides a clear, standard format for function documentation improving usability and maintainability.

Task 5: Review and Correct Docstrings

Prompt: "Identify and fix inaccuracies in existing docstrings."

Code:

```

1 def factorial(n: int) -> int:
2     """
3     Incorrect docstring: Returns the sum of numbers up to n.
4
5     Args:
6     |     n (int): A non-negative integer.
7
8     Returns:
9     |     int: Incorrect description.
10    """
11    result = 1
12    for i in range(1, n + 1):
13        result *= i
14    return result
15
16 # Corrected Docstring
17
18 def corrected_factorial(n: int) -> int:
19     """
20     Calculate the factorial of a non-negative integer.
21
22     Args:
23     |     n (int): Non-negative integer.
24
25     Returns:
26     |     int: Factorial of n.
27
28     Example:
29     |     >>> corrected_factorial(5)
30     |     120
31     """
32    result = 1
33    for i in range(1, n + 1):
34        result *= i
35    return result
36
37 if __name__ == "__main__":
38     print("Task 5 - Factorial of 5:", corrected_factorial(5))
39     """

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Task 5 - Factorial of 5: 120
PS D:\vscode\puth>

```

Observation: The corrected docstring now accurately reflects the function's behavior following Google style.

Task 6: Prompt Comparison Experiment

Prompt: - Vague: "Add comments to this function." - Detailed: "Add a Google-style docstring with description, parameters, return type, and example usage."

Code:

```

email_validator.py > ...
1  def cube(x: int) -> int:
2      return x * x * x
3
4  def cube_vague(x: int) -> int:
5      # Multiply x by itself three times
6      return x * x * x
7
8  def cube_detailed(x: int) -> int:
9      """
10     Calculate the cube of an integer.
11
12     Args:
13         x (int): The number to cube.
14
15     Returns:
16         int: The cube of x.
17
18     Example:
19         >>> cube_detailed(3)
20         27
21     """
22     return x * x * x
23
24  if __name__ == "__main__":
25      print("Vague prompt output (cube of 3):", cube_vague(3))
26      print("Detailed prompt output (cube of 3):", cube_detailed(3))
27  """

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Vague prompt output (cube of 3): 27
Detailed prompt output (cube of 3): 27
PS D:\vscode\puth>

```

Output:

Vague prompt output (cube of 3): 27
Detailed prompt output (cube of 3): 27

Observation: The detailed prompt produces a complete Google-style docstring improving function usability, while the vague prompt produces only a simple inline comment.