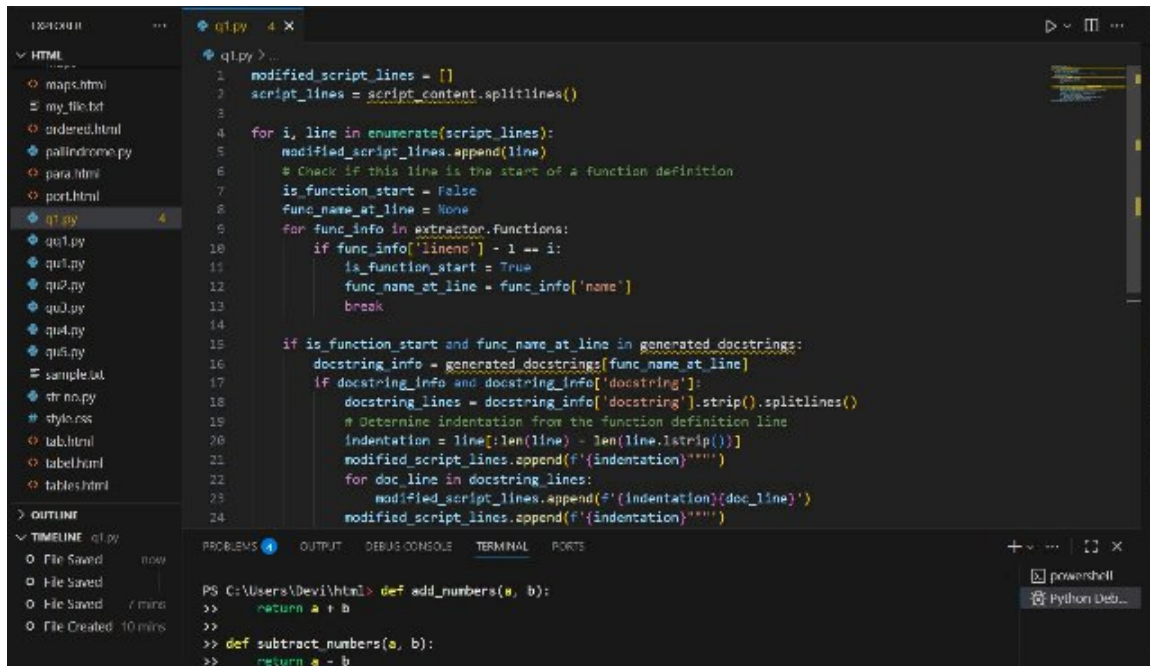


# ASSIGNMENT 9.2

2403A52049

BATCH-3

TASK-1:



```
1 modified_script_lines = []
2 script_lines = script_content.splitlines()
3
4 for i, line in enumerate(script_lines):
5     modified_script_lines.append(line)
6     # Check if this line is the start of a function definition
7     is_function_start = False
8     func_name_at_line = None
9     for func_info in extractor.functions:
10        if func_info['lineno'] - 1 == i:
11            is_function_start = True
12            func_name_at_line = func_info['name']
13            break
14
15    if is_function_start and func_name_at_line in generated_docstrings:
16        docstring_info = generated_docstrings[func_name_at_line]
17        if docstring_info and docstring_info['docstring']:
18            docstring_lines = docstring_info['docstring'].strip().splitlines()
19            # Determine indentation from the function definition line
20            indentation = line[:len(line) - len(line.lstrip())]
21            modified_script_lines.append(f'{indentation}"""')
22            for doc_line in docstring_lines:
23                modified_script_lines.append(f'{indentation}{doc_line}')
24            modified_script_lines.append(f'{indentation}"""')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Devil\html> def add_numbers(a, b):
>>     return a + b
>>
>> def subtract_numbers(a, b):
>>     return a - b
```

```
6 # Check if this line is the start of a function definition
7 is_function_start = False
8 func_name_at_line = None
9 for func_info in extractor.functions:
10     if func_info['lineno'] - 1 == i:
11         is_function_start = True
12         func_name_at_line = func_info['name']
13         break
14
15 if is_function_start and func_name_at_line in generated_docstrings:
16     docstring_info = generated_docstrings[func_name_at_line]
17     if docstring_info and docstring_info['docstring']:
18         docstring_lines = docstring_info['docstring'].strip().splitlines()
19         # Determine indentation from the function definition line
20         indentation = line[:len(line) - len(line.lstrip())]
21         modified_script_lines.append(f'{indentation}"""')
22         for doc_line in docstring_lines:
23             modified_script_lines.append(f'{indentation}{doc_line}')
24         modified_script_lines.append(f'{indentation}"""')
25 modified_script_content = "\n".join(modified_script_lines)
26 print(modified_script_content)
```

PS C:\Users\Devi\html> def add\_numbers(a, b):  
>> return a + b  
>>  
>> def subtract\_numbers(a, b):  
>> return a - b

TAS

K-2:

```
1 def find_longest_substring(s):
2     max_len = 0
3     start = 0
4     used_chars = {}
5
6     for i, char in enumerate(s):
7         # If the character is already in the dictionary and its index is within the current window
8         if char in used_chars and start <= used_chars[char]:
9             # Move the start to one position after the last occurrence of the current character
10            start = used_chars[char] + 1
11        else:
12            # Update max_len if we found a longer substring without repeating characters
13            max_len = max(max_len, i - start + 1)
14            # Store/Update the index of the current character
15            used_chars[char] = i
16
17    return max_len
```

PS C:\Users\sathw & C:\Users\sathw\anaconda3\python.exe c:/Users/sathw/OneDrive/Desktop/HTML/9.2.py  
PS C:\Users\sathw & C:\Users\sathw\anaconda3\python.exe c:/Users/sathw/OneDrive/Desktop/HTML/9.2.py  
PS C:\Users\sathw> 23  
PS C:\Users\sathw>

TASK-3:

The screenshot shows the Visual Studio Code editor with a Python file named `q4.py` open. The file contains the following code:

```
1 import re
2
3 def convert_inline_comments_to_docstrings_reduced(script_content: str) -> str:
4     """
5     Transforms inline comments into structured Google-style docstrings.
6
7     This version is a streamlined implementation that identifies and converts
8     inline comments immediately following a function definition into a single,
9     well-formatted docstring. It assures a predictable comment structure.
10
11     Args:
12         script_content (str): The script as a string.
13
14     Returns:
15         str: The modified script with new docstrings.
16     """
17     lines = script_content.splitlines()
18     modified_lines = []
19     i = 0
20
21     while i < len(lines):
22         line = lines[i]
23         func_match = re.match(r'(\s*)def\s+', line)
24         modified_lines.append(line)
```

The Explorer sidebar on the left shows a list of files, including `q1.py` through `q11.py`. The Timeline panel at the bottom shows file creation and saving events. The Output panel at the bottom right shows the output of the function, which is a docstring for the `convert_inline_comments_to_docstrings_reduced` function.

The screenshot shows the Visual Studio Code editor with the same Python file `q4.py` open. The code has been updated to include the logic for processing the comments:

```
25
26     line = lines[i]
27     func_match = re.match(r'(\s*)def\s+', line)
28     modified_lines.append(line)
29
30     if func_match:
31         indentation = func_match.group(1)
32         comments = []
33         j = i + 1
34         while j < len(lines) and lines[j].strip().startswith('#'):
35             comments.append(lines[j].strip('#').strip())
36             j += 1
37
38         if comments:
39             # Use a single string join for the body of the docstring
40             docstring_body = "\n".join(f"{indentation} {c}" for c in comments)
41             docstring = f'"""{docstring_body}\n{indentation}"""'
42             modified_lines.append(f'{indentation} {docstring}')
43             i = j # Skip the processed comment lines
44             continue
45
46     i += 1
47
48     return "\n".join(modified_lines)
49
50 # --- Example Usage ---
```

The Explorer sidebar and Timeline panel are the same as in the previous screenshot. The Output panel at the bottom right shows the output of the function, which is a docstring for the `convert_inline_comments_to_docstrings_reduced` function.

OUTPUT:

```
def calculate_area(length, width):
    """
    # This function calculates the area of a rectangle.
    # Args:
    #   length (float): The length of the rectangle.
    #   width (float): The width of the rectangle.
    # Returns:
    #   float: The calculated area.
    """
    return length * width
PS C:\Users\Devi\html> 
```

## TASK-4:

```
1 def convert_comments_to_docstring():
2     print("Write your function code with inline comments (and input with a black line):")
3     lines = []
4     while True:
5         line = input()
6         if line.strip() == "":
7             break
8         lines.append(line)
9
10    func_lines = []
11    comments = []
12    for line in lines:
13        stripped = line.strip()
14        if stripped.startswith("#"):
15            comments.append(stripped.lstrip("#").strip())
16        else:
17            func_lines.append(line)
18
19    docstring = """
20    for comment in comments:
21        docstring += f"    {comment}\n"
22    docstring += """
23
24    # Insert docstring after function definition
25    for i, line in enumerate(func_lines):
26        if line.strip().startswith("def "):
27            func_lines.insert(i + 1, docstring)
28            break
29
30    result = "\n".join(func_lines)
31    print("\nTransformed function:\n")
32    print(result)
33    return result
34
35 if __name__ == "__main__":
36     convert_comments_to_docstring()
```

```
1 def convert_comments_to_docstring():
2     print("Write your function code with inline comments (and input with a black line):")
3     lines = []
4     while True:
5         line = input()
6         if line.strip() == "":
7             break
8         lines.append(line)
9
10    func_lines = []
11    comments = []
12    for line in lines:
13        stripped = line.strip()
14        if stripped.startswith("#"):
15            comments.append(stripped.lstrip("#").strip())
16        else:
17            func_lines.append(line)
18
19    docstring = """
20    for comment in comments:
21        docstring += f"    {comment}\n"
22    docstring += """
23
24    # Insert docstring after function definition
25    for i, line in enumerate(func_lines):
26        if line.strip().startswith("def "):
27            func_lines.insert(i + 1, docstring)
28            break
29
30    result = "\n".join(func_lines)
31    print("\nTransformed function:\n")
32    print(result)
33    return result
34
35 if __name__ == "__main__":
36     convert_comments_to_docstring()
```

## TASK-5:

```

1  import ast
2
3  def get_connected_doctring(old doctring):
4      print("Existing doctstring:")
5      print(old doctring)
6      print("Let's make the corrected doctstring (leave blank to keep unchanged):")
7      new doctring = input()
8      return new doctring if new doctstring.strip() else old doctstring
9
10 def correct_doctrings_in_file(filename):
11     with open(filename, 'r', encoding='utf 8') as f:
12         source = f.read()
13
14         tree = ast.parse(source)
15         edits = []
16
17     for node in ast.walk(tree):
18         if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef, ast.ClassDef, ast.Module)):
19             old doctring = ast.get doctring(node)
20             if old doctstring is not None:
21                 new doctring = get_connected_doctring(old doctring)
22                 if new doctring != old doctstring:
23                     # Edit the doctring node
24                     doct_node = node.body[0]
25                     if isinstance(doct_node, ast.Expr) and isinstance(doct_node.value, ast.Str):

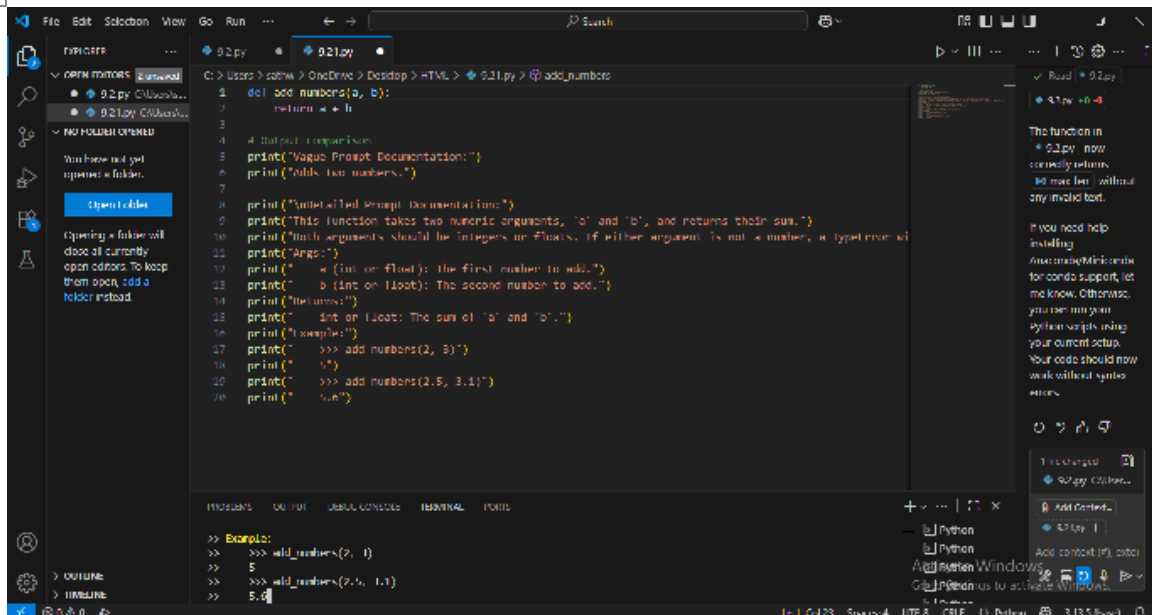
```

```

26 def correct_doctrings_in_file(filename):
27     # Iterate through the source code and find doctstrings
28     edits.append((doc node.filename, new doctring))
29
30 # Apply edits to the source code
31 lines = source.split lines()
32 for filename, new doctring in sorted(edits, reverse=True):
33     # Replace the doctstring in lines
34     lines[filename - 1] = f"""{new doctring}"""
35
36 with open(filename, 'w', encoding='utf 8') as f:
37     f.write('\n'.join(lines))
38
39 if __name__ == "__main__":
40     filename = input("Enter the Python file path to review doctstrings: ")
41     correct_doctrings_in_file(filename)
42     print("Doctring review and correction complete.")

```

## TASK-6:



## OUTPUT:

a (int or float): The first number to add.

b (int or float): The second number to add.

Returns:

int or float: The sum of `a` and `b`.

Example:

```
>>> add_numbers(2, 3)
```

```
5
```

```
>>> add_numbers(2.5, 3.1)
```

```
5.6
```