

Lab assignment – 9.4

Task 1: Auto-Generating Function Documentation in a Shared Codebase

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The main editor tab displays a Python file named `ai lab 9.4.py`. The code contains two functions: `add` and `is_even`. Both functions have detailed docstrings with descriptions, arguments, and examples. The `add` function adds two integers and returns the result. The `is_even` function checks if a number is even and returns a boolean value. The code is annotated with comments and triple quotes for the docstrings.

```
1  def add(a: int, b: int) -> int:
2      """
3          Adds two integers and returns the result.
4      """
5      Args:
6          a (int): First integer.
7          b (int): Second integer.
8
9      Returns:
10         int: Sum of a and b.
11
12     Example:
13     >>> add(3, 4)
14     7
15     """
16
17     return a + b
18
19
20 def is_even(num: int) -> bool:
21     """
22         Checks whether a number is even.
23     """
24     Args:
25         num (int): The number to check.
26
27     Returns:
28         bool: True if the number is even, otherwise False.
29
30     Example:
31     >>> is_even(10)
32     True
33     """
34
35     return num % 2 == 0
36
```

Below the editor, the terminal window shows the command `python.exe "D:\AI Coding\ai lab 9.4.py"` being run. The status bar at the bottom indicates the current workspace is `D:\AI Coding`.

Task 2: Enhancing Readability Through AI-Generated Inline Comments

The screenshot shows a code editor interface with a dark theme. The main window displays a Python file named `ai lab 9.4.py`. The code defines a `TokenBucket` class with methods for initializing capacity and fill rate, calculating tokens available, and allowing requests. The code includes several AI-generated inline comments explaining the logic and avoiding constant updates.

```
1 import time
2
3 class TokenBucket:
4     def __init__(self, capacity, fill_rate):
5         self.capacity = capacity
6         self.tokens = capacity
7         self.fill_rate = fill_rate
8         self.last_filled = time.monotonic()
9
10    def allow_request(self, tokens_requested):
11        now = time.monotonic()
12
13        # Calculate how much the bucket has filled since the last check.
14        # This "lazy refill" avoids needing a background thread to constantly update the bucket.
15        added_tokens = (now - self.last_filled) * self.fill_rate
16
17        # Ensure the bucket doesn't overflow its capacity if there's a long period of inactivity.
18        self.tokens = min(self.capacity, self.tokens + added_tokens)
19        self.last_filled = now
20
21        if self.tokens >= tokens_requested:
22            self.tokens -= tokens_requested
23            return True
24
25        return False
26
27 # Example Usage
28 limiter = TokenBucket(capacity=5, fill_rate=1) # 5 tokens max, 1 added per second
29
30 for i in range(8):
31     if limiter.allow_request():
32         print("Request {}:{}: Allowed".format(i+1))
33     else:
34         print("Request {}:{}: Rate limited".format(i+1))
35     time.sleep(0.5)
```

Below the code editor is a terminal window showing the execution of the script. The output shows 8 requests, each labeled "Allowed".

```
PS D:\AI Coding & C:/Users/MDALI/AppData/Local/Programs/Python/Python311/python.exe "d:/AI Coding/ai lab 9.4.py"
PS D:\AI Coding & C:/Users/MDALI/AppData/Local/Programs/Python/Python311/python.exe "d:/AI Coding/ai lab 9.4.py"
PS D:\AI Coding & C:/Users/MDALI/AppData/Local/Programs/Python/Python311/python.exe "d:/AI Coding/ai lab 9.4.py"
Request 1: Allowed
Request 2: Allowed
Request 3: Allowed
Request 4: Allowed
Request 5: Allowed
Request 6: Allowed
Request 7: Allowed
Request 8: Allowed
```

A status bar at the bottom right indicates "Extension Bisect is active" and provides a link to "I can't find it".

Task 3: Generating Module-Level Documentation for a Python Package

```

1  # Pathfinding and Grid Navigation Utility
2
3  # This module provides efficient algorithms for calculating the shortest path
4  # between nodes in a 2D weighted grid environment.
5
6  Dependencies:
7      - math (Standard Library): used for the priority queue implementation.
8      - math (Standard Library): used for distance calculations.
9
10 Key Functions:
11     - calculate_heuristic: Calculates the cost from a node to the target.
12     - find_shortest_path: Returns the shortest path from start to goal.
13
14 Example Usage:
15
16     >>> grid = [(0, 0, 0), (0, 1, 0), (0, 0, 0)]
17     >>> start = (0, 0)
18     >>> goal = (2, 2)
19     >>> path = find_shortest_path(grid, start, goal)
20
21     print(path)
22
23     [(0, 0), (0, 1), (1, 1), (2, 1), (2, 2)]
24
25
26 import heapq
27 import math
28
29 def calculate_heuristic(current, goal):
30
31     # Calculates the Euclidean distance between two points.
32
33     Args:
34         current (tuple): (x, y) coordinates of the current node.
35         goal (tuple): (x, y) coordinates of the destination.
36
37     Returns:
38         float: The straight-line distance to the goal.
39
40     return math.sqrt((current[0] - goal[0])**2 + (current[1] - goal[1])**2)
41
42 def find_shortest_path(grid, start, goal):
43
44     # Finds the optimal path through a grid using the A* algorithm.
45
46     Args:
47         grid (list[list[int]]): 2D grid where 0 is traversable and 1 is a wall.
48         start (tuple): (x, y) coordinates of the start node.
49         goal (tuple): (x, y) coordinates.
50
51     Returns:
52         list[tuple] or None: The shortest path as a list of coordinates,
53         or None if no path exists.
54
55     neighbors = [(0, 1), (0, -1), (1, 0), (-1, 0)]
56
57
58 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Standard Deviation: 5.46
Z-Scores: [-0.29309732217892365, 0.622831809630216, -1.575398106711719, 0.9892034623538718, 0.2564601569065601]
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab 9.4.py"
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"

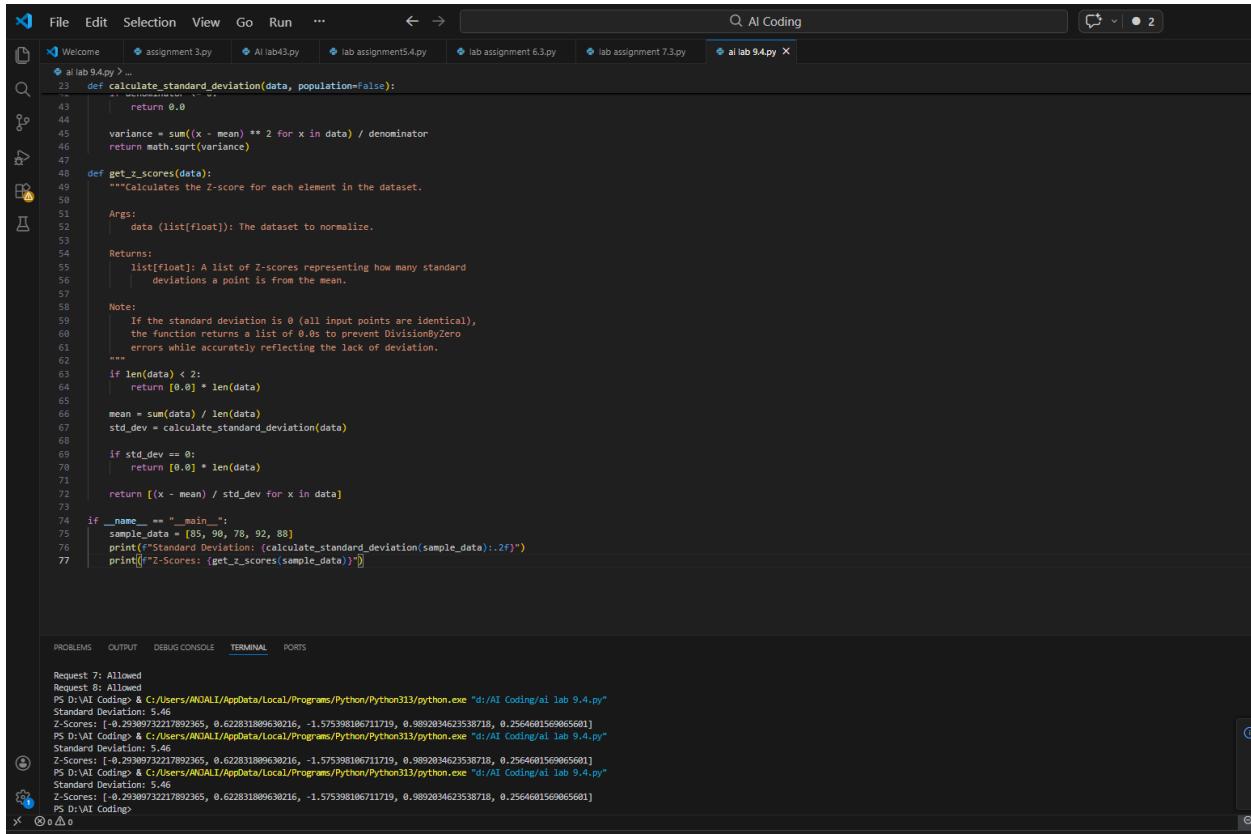
```

1  #!/usr/bin/env python3
2
3  def find_shortest_path(grid, start, goal):
4
5      open_set = []
6      came_from = {}
7      g_score = {node: float('inf') for node in grid}
8      f_score = {node: float('inf') for node in grid}
9
10     g_score[start] = 0
11     f_score[start] = g_score[start] + calculate_heuristic(start, goal)
12
13     heapq.heappush(open_set, (f_score[start], start))
14
15     while open_set:
16
17         current = heapq.heappop(open_set)[1]
18
19         if current == goal:
20             path = []
21             while current in came_from:
22                 path.append(current)
23                 current = came_from[current]
24             path.append(start)
25             path.reverse()
26             return path
27
28         for dx, dy in neighbors:
29             neighbor = (current[0] + dx, current[1] + dy)
30
31             # Boundary and collision check
32             if 0 <= neighbor[0] < len(grid) and 0 <= neighbor[1] < len(grid[0]):
33                 if grid[neighbor[0]][neighbor[1]] == 1:
34                     continue
35
36                 tentative_g_score = g_score[current] + 1
37
38                 # If this path to neighbor is better than any previous one, record it.
39                 if tentative_g_score < g_score.get(neighbor, float('inf')):
40                     came_from[neighbor] = current
41                     g_score[neighbor] = tentative_g_score
42
43                     f_score[neighbor] = g_score[neighbor] + calculate_heuristic(neighbor, goal)
44
45                     heapq.heappush(open_set, (f_score[neighbor], neighbor))
46
47     return None
48
49
50 if __name__ == "__main__":
51     # 0 = Path, 1 = Wall
52     test_grid = [
53         [0, 0, 0, 0],
54         [1, 0, 0, 1],
55         [0, 0, 0, 0],
56         [0, 1, 1, 0]
57     ]
58
59     print(f"Path: {find_shortest_path(test_grid, (0, 0), (3, 3))}")
60
61
62 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Standard Deviation: 5.46
Z-Scores: [-0.29309732217892365, 0.622831809630216, -1.575398106711719, 0.9892034623538718, 0.2564601569065601]
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
Standard Deviation: 5.46
Z-Scores: [-0.29309732217892365, 0.622831809630216, -1.575398106711719, 0.9892034623538718, 0.2564601569065601]
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
Usage: python scaffold.py <target_file.py>
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
Usage: python scaffold.py <target_file.py>
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (3, 3)]
PS D:\AI Coding &

Task 4: Converting Developer Comments into Structured Docstrings



The screenshot shows a code editor with a Python script named `alab94.py`. The code defines two functions: `calculate_standard_deviation` and `get_z_scores`. The `get_z_scores` function includes detailed docstrings with arguments, return types, and notes about standard deviation zero.

```
def calculate_standard_deviation(data, population=False):
    ...
    return 0.0

def get_z_scores(data):
    """Calculates the Z-score for each element in the dataset.

    Args:
        data (list[float]): The dataset to normalize.

    Returns:
        list[float]: A list of Z-scores representing how many standard
                     deviations a point is from the mean.

    Note:
        If the standard deviation is 0 (all input points are identical),
        the function returns a list of 0.0s to prevent DivisionByZero
        errors while accurately reflecting the lack of deviation.

    """
    if len(data) < 2:
        return [0.0] * len(data)

    mean = sum(data) / len(data)
    std_dev = calculate_standard_deviation(data)

    if std_dev == 0:
        return [0.0] * len(data)

    return [(x - mean) / std_dev for x in data]

if __name__ == "__main__":
    sample_data = [85, 90, 78, 92, 88]
    print(f"Standard Deviation: {calculate_standard_deviation(sample_data):.2f}")
    print(f"Z-Scores: {get_z_scores(sample_data)}")
```

The terminal tab at the bottom shows the execution of the script and its output:

```
Request 7: Allowed
Request 8: Allowed
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python313/python.exe "d:\AI Coding\al lab 9.4.py"
Standard Deviation: 5.45
Z-Scores: [-0.29399732217892365, 0.62283188630216, -1.575398106711719, 0.9892934623538718, 0.2564601569065601]
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python313/python.exe "d:\AI Coding\al lab 9.4.py"
Standard Deviation: 5.46
Z-Scores: [-0.29399732217892365, 0.62283188630216, -1.575398106711719, 0.9892934623538718, 0.2564601569065601]
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python313/python.exe "d:\AI Coding\al lab 9.4.py"
Standard Deviation: 5.46
Z-Scores: [-0.29399732217892365, 0.62283188630216, -1.575398106711719, 0.9892934623538718, 0.2564601569065601]
PS D:\AI Coding
```

Task 5: Building a Mini Automatic Documentation Generator

The screenshot shows two instances of the PyCharm IDE. The top instance displays the `target_file.py` file, which contains Python code for generating Google-style docstrings. The bottom instance shows the terminal output of running the script, demonstrating its functionality on a sample input file.

```

File Edit Selection View Go Run ... ← → ⌂ AI Coding
Welcome target_file.py Al assignment 3.py Al lab8.py Al assignment5.py Al assignment 6.3.py Al assignment 7.3.py target_file.py
1
2 Docstring Scaffolding Utility
3
4 This module provides a tool to automatically insert Google-style docstring
5 placeholders into Python source files that lack documentation.
6
7 Dependencies:
8   - ast (Standard Library): Used to parse and traverse the Python code structure.
9   - sys (Standard Library): Used for command-line argument handling.
10
11 Key Functions:
12   generate_scaffold: Processes source code to find and document nodes.
13   main: Handles file I/O and command-line execution.
14
15
16 import ast
17 import sys
18
19 def generate_scaffold(source_code):
20     """Parses Python source and inserts Google-style docstring placeholders.
21
22     Args:
23         source_code (str): The raw string content of a .py file.
24
25     Returns:
26         str: The modified source code with docstring templates inserted.
27
28     Example:
29         >>> code = "def add(a, b): return a + b"
30         >>> print(generate_scaffold(code))
31         >>> def add(a, b):
32             """
33                 \\""\"
34                     Args:
35                         a (type): Description.
36
37                         b (type): Description.
38
39                     ...
40             """
41             return a + b
42
43     try:
44         tree = ast.parse(source_code)
45     except SyntaxError:
46         return source_code
47
48     lines = source_code.splitlines()
49
50     # Identify functions and classes.
51     nodes = [n for n in ast.walk(tree) if isinstance(n, (ast.FunctionDef, ast.ClassDef, ast.AsyncFunctionDef))]
52
53     # We process nodes in reverse order of their line numbers.
54     # This is vital because inserting text shifts the line numbers of everything
55     # below the insertion point. By starting at the bottom, we preserve the
56     # coordinate system for the nodes above.
57     # ...
58     # ...
59
60     PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
61 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\ai_lab_9.4.py"
62 Usage: python scaffold.py target_file.py
63 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
64 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
65 Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (3, 3)]
66 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
67 Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (3, 3)]
68 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
④ File Edit Selection View Go Run ... ← → ⌂ AI Coding
Welcome target_file.py Al assignment 3.py Al lab8.py Al assignment 5.py Al assignment 6.3.py Al assignment 7.3.py target_file.py
1
2 def generate_scaffold(source_code):
3     # below the insertion point: By starting at the bottom, we preserve the
4     # coordinate system for the nodes above.
5     # ...
58     # ...
59
60     for node in nodes:
61         # skip nodes that already have documentation to avoid duplication.
62         if ast.get_docstring(node):
63             continue
64
65         # Use col_offset to determine the exact indentation level.
66         # This ensures the docstring aligns perfectly with the function body.
67         indent = '    ' * node.col_offset
68         inner_indent = indent + '    '
69
70         if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef)):
71             # Function or class definition: The docstring is the first argument.
72             args = [arg for arg in node.args.args if arg.arg not in ('self', 'cls')]
73             args_block = '\n'.join(['        ' * inner_indent + (type(arg).__name__ + ': ' + arg.description) for arg in args])
74
75             doc = f'{inner_indent}'''Summary of function.\n{inner_indent}{args_block}{inner_indent}'''Type description.\n{inner_indent}'''
76
77         else:
78             doc = f'{inner_indent}'''Summary of class.\n{inner_indent}{node.attributes}{inner_indent}'''Attr (type): Description.\n{inner_indent}'''
79
80         # node.line is 1-indexed; inserting at this index puts text on the line
81         # immediately following the 'def' or 'class' statement.
82         lines.insert(node.line, doc)
83
84     return '\n'.join(lines)
85
86     def main(filename):
87         """Reads a file and writes the scaffolded version back to disk.
88
89         Args:
90             filename (str): Path to the .py file to be processed.
91
92         Returns:
93             None
94
95         with open(filename, 'r', encoding='utf-8') as f:
96             content = f.read()
97
98             scaffolded = generate_scaffold(content)
99
100            with open(filename, 'w', encoding='utf-8') as f:
101                f.write(scaffolded)
102
103            print(f"Successfully added placeholders to {filename}")
104
105
106    PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
107 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\ai_lab_9.4.py"
108 Usage: python scaffold.py target_file.py
109 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
110 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
111 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
112 Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (3, 3)]
113 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
114 PS D:\AI Coding & C:\Users\MOULI\Apidata\local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"

```