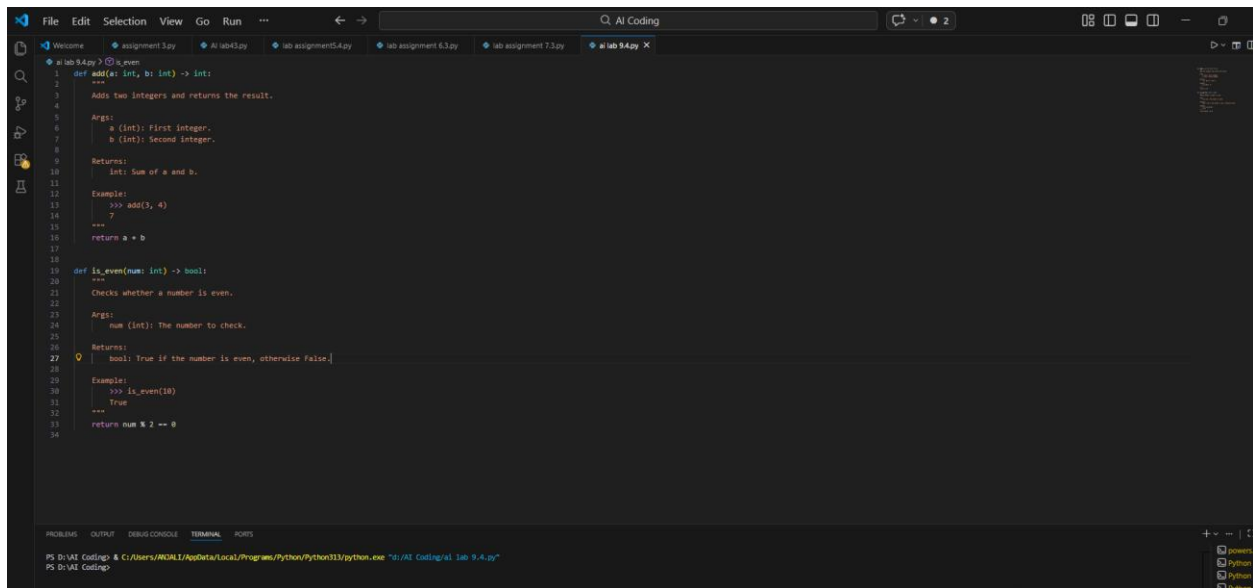


## Lab assignment – 9.4

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



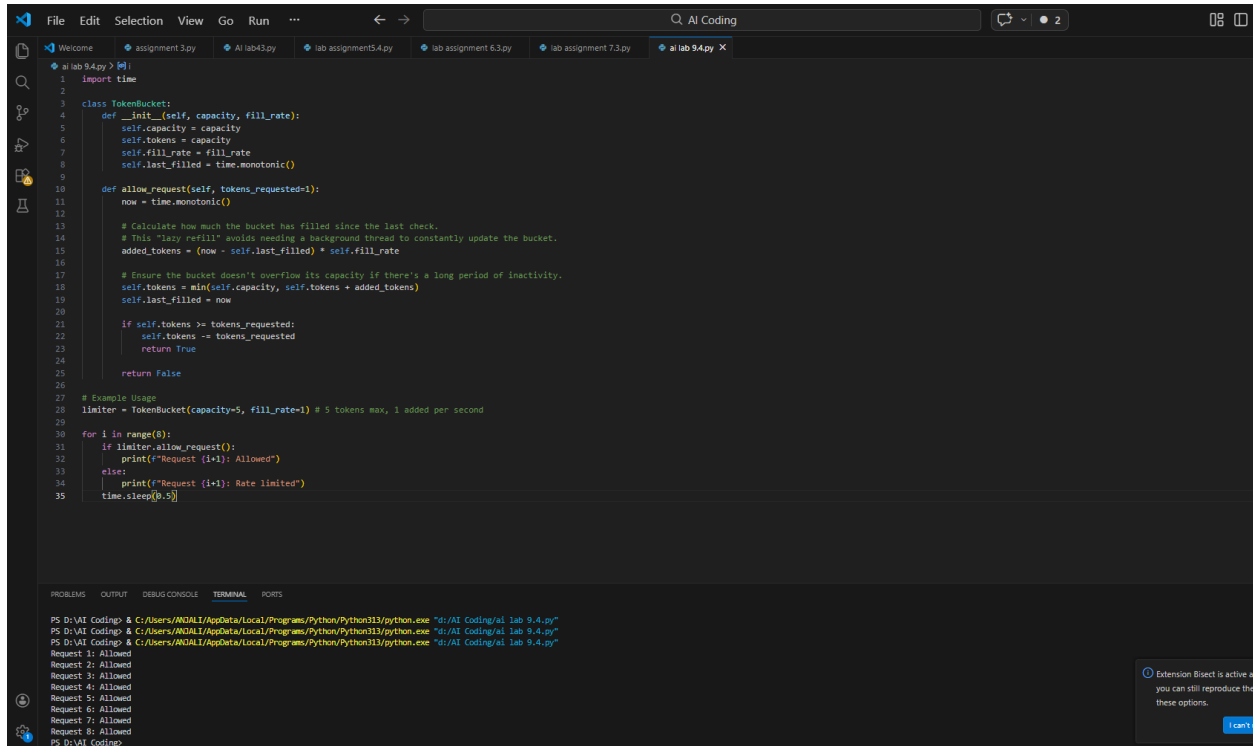
```
1 def add(a: int, b: int) -> int:
2     """
3     Adds two integers and returns the result.
4
5     Args:
6         a (int): First integer.
7         b (int): Second integer.
8
9     Returns:
10        int: Sum of a and b.
11
12     Example:
13         >>> add(3, 4)
14         7
15     """
16   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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\AI Coding> C:\Users\MOALI\AppData\Local\Programs\Python\Python311\python.exe "D:\AI Coding\ai\_lab\_9.4.py"

PS D:\AI Coding>

## Task 2: Enhancing Readability Through AI-Generated Inline Comments



```
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=1):
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        return False
25
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(f"Request {i+1}: Allowed")
33     else:
34         print(f"Request {i+1}: Rate limited")
35         time.sleep(0.5)
```

PS D:\AI Coding> & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/ai\_lab\_9.4.py"

PS D:\AI Coding> & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/ai\_lab\_9.4.py"

PS D:\AI Coding> & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\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

PS D:\AI Coding>

Extension Bisect is active as you can still reproduce the these options.

Can't

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

```
File Edit Selection View Go Run ... AI Coding
target_file.py > find_shortest_path

"""
Pathfinding and Grid Navigation Utility
"""
This module provides efficient algorithms for calculating the shortest path
between nodes in a 2D weighted grid environment.

Dependencies:
- heapq (Standard Library): Used for the priority queue implementation.
- math (Standard Library): Used for distance calculations.

Key Functions:
- calculate_heuristic: Estimates the cost from a node to the target.
- find_shortest_path: Executes the A* search algorithm.

Example Usage:
"""
grid = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
start = (0, 0)
goal = (2, 2)
path = find_shortest_path(grid, start, goal)
print(path)

"""
[[0, 0], (0, 1), (1, 1), (2, 1), (2, 2)]

import heapq
import math

def calculate_heuristic(current, goal):
    """
    Calculates the Euclidean distance between two points.

    Args:
        current (tuple): (x, y) coordinates of the current node.
        goal (tuple): (x, y) coordinates of the destination.

    Returns:
        float: The straight-line distance to the goal.
    """
    return math.sqrt((current[0] - goal[0])**2 + (current[1] - goal[1])**2)

def find_shortest_path(grid, start, goal):
    """
    Finds the optimal path through a grid using the A* algorithm.

    Args:
        grid (list[list[int]]): 2D array where 0 is traversable and 1 is a wall.
        start (tuple): Starting (x, y) coordinates.
        goal (tuple): Target (x, y) coordinates.

    Returns:
        list[tuple] or None: The shortest path as a list of coordinates,
        or None if no path exists.
    """
    neighbors = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    """
Standard Deviation: 5.46
Z-Scores: [-0.29389732217892365, 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\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\ & C:\Users\ANDALI\AppData\Local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
"""
```

```
File Edit Selection View Go Run ... AI Coding
target_file.py > ...

def find_shortest_path(grid, start, goal):
    """
    Finds the optimal path through a grid using the A* algorithm.

    Args:
        grid (list[list[int]]): 2D array where 0 is traversable and 1 is a wall.
        start (tuple): Starting (x, y) coordinates.
        goal (tuple): Target (x, y) coordinates.

    Returns:
        list[tuple] or None: The shortest path as a list of coordinates,
        or None if no path exists.
    """
    neighbors = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    # Initial setup
    current = start
    path = [current]
    return path[:-1]

    for dx, dy in neighbors:
        neighbor = (current[0] + dx, current[1] + dy)

        # Boundary and collision check
        if 0 <= neighbor[0] < len(grid) and 0 <= neighbor[1] < len(grid[0]):
            if grid[neighbor[0]][neighbor[1]] == 1:
                continue

            tentative_g_score = g_score[current] + 1

            # If this path to neighbor is better than any previous one, record it.
            if tentative_g_score < g_score.get(neighbor, float('inf')):
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score

            # f_score = g_score (cost to reach) + h_score (estimated cost to goal).
            # This balance allows A* to be 'informed' and 'greedy' simultaneously.
            f_score = tentative_g_score + calculate_heuristic(neighbor, goal)
            heapq.heappush(open_set, (f_score, neighbor))

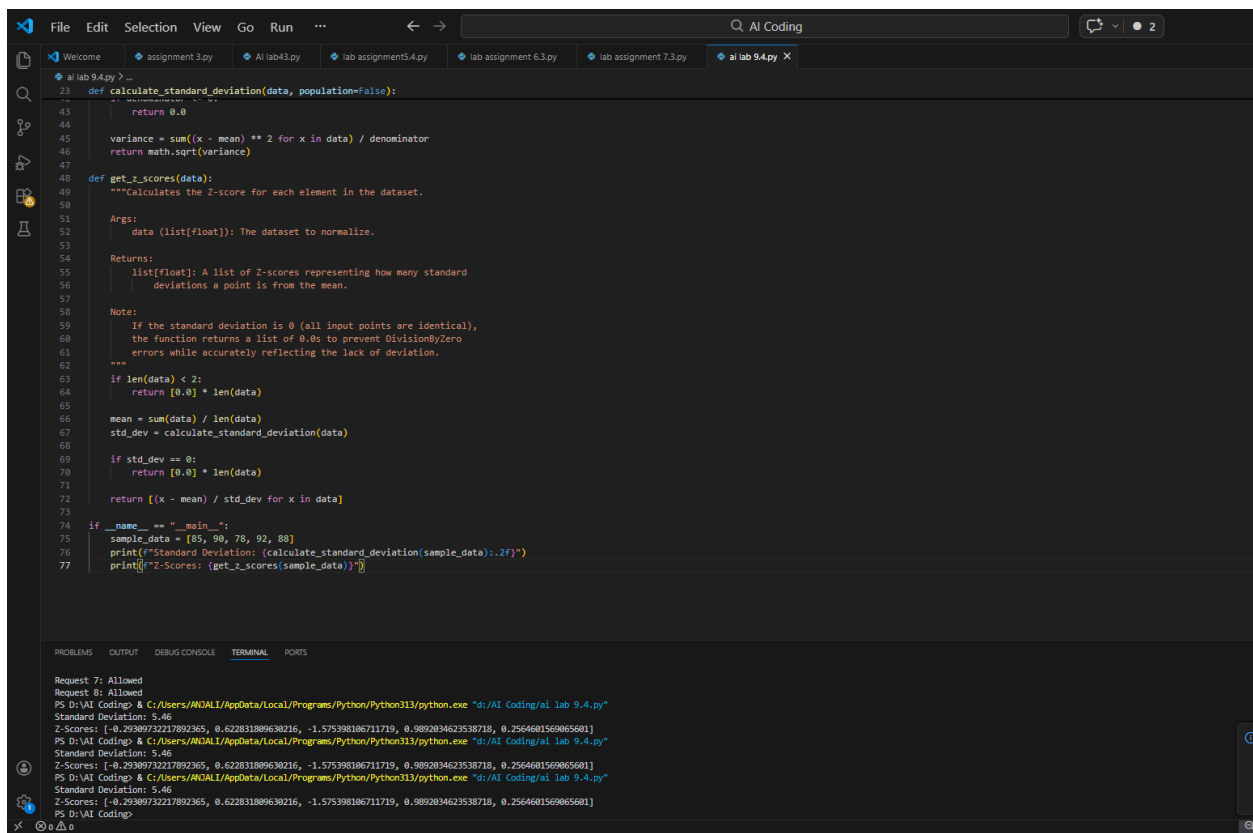
    return None

if __name__ == "__main__":
    # 0 = Path, 1 = Wall
    test_grid = [
        [0, 0, 0, 0],
        [1, 1, 0, 1],
        [0, 0, 0, 0],
        [0, 1, 1, 0]
    ]
    print(f"Path: {find_shortest_path(test_grid, (0, 0), (3, 3))}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Standard Deviation: 5.46
Z-Scores: [-0.29389732217892365, 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: [-0.29389732217892365, 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"
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\ & C:\Users\ANDALI\AppData\Local\Programs\Python\Python313\python.exe "d:\AI Coding\target_file.py"
```

## Task 4: Converting Developer Comments into Structured Docstrings



```
23 def calculate_standard_deviation(data, population=False):
24     """Calculates the standard deviation of a dataset.
25     """
26     return 0.0
27
28 variance = sum((x - mean) ** 2 for x in data) / denominator
29 return math.sqrt(variance)
30
31 def get_z_scores(data):
32     """Calculates the Z-score for each element in the dataset.
33     """
34     Args:
35         data (list[float]): The dataset to normalize.
36
37     Returns:
38         list[float]: A list of Z-scores representing how many standard
39         deviations a point is from the mean.
40
41     Note:
42         If the standard deviation is 0 (all input points are identical),
43         the function returns a list of 0.0s to prevent DivisionByZero
44         errors while accurately reflecting the lack of deviation.
45     """
46     if len(data) < 2:
47         return [0.0] * len(data)
48
49     mean = sum(data) / len(data)
50     std_dev = calculate_standard_deviation(data)
51
52     if std_dev == 0:
53         return [0.0] * len(data)
54
55     return [(x - mean) / std_dev for x in data]
56
57 if __name__ == "__main__":
58     sample_data = [85, 90, 78, 92, 88]
59     print(f"Standard Deviation: {calculate_standard_deviation(sample_data):.2f}")
60     print(f"Z-Scores: {get_z_scores(sample_data)}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Request 7: Allowed  
Request 8: Allowed  
PS D:\AI Coding & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/ai lab 9.4.py"  
Standard Deviation: 5.46  
Z-Scores: [-0.29389732217892365, 0.622831809630216, -1.575398186711719, 0.9892034623538718, 0.2564601569065601]  
PS D:\AI Coding & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/ai lab 9.4.py"  
Standard Deviation: 5.46  
Z-Scores: [-0.29389732217892365, 0.622831809630216, -1.575398186711719, 0.9892034623538718, 0.2564601569065601]  
PS D:\AI Coding & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/ai lab 9.4.py"  
Standard Deviation: 5.46  
Z-Scores: [-0.29389732217892365, 0.622831809630216, -1.575398186711719, 0.9892034623538718, 0.2564601569065601]  
PS D:\AI Coding

## Task 5: Building a Mini Automatic Documentation Generator

The image shows a VS Code editor with a Python file named `target_file.py`. The file contains a docstring scaffolding utility. The script takes a source code string as input and inserts docstrings. It includes comments in Spanish explaining its purpose and usage. The script is being executed in a terminal window at the bottom, showing the command `python scaffolder.py target_file.py` and its output, which is a docstring for a function named `add`. The terminal output shows the docstring being inserted into the source code file.

```

1  """
2  Docstring Scaffolding Utility
3  =====
4
5  This module provides a tool to automatically insert Google-style docstring
6  placeholders into Python source files that lack documentation.
7
8  Dependencies:
9  - ast (Standard Library): Used to parse and traverse the Python code structure.
10 - sys (Standard Library): Used for command-line argument handling.
11
12 Key Functions:
13 - generate_scaffold: Processes source code to find and document nodes.
14 - main: Handles file I/O and command-line execution.
15 """
16
17 import ast
18 import sys
19
20 def generate_scaffold(source_code):
21     """Parses Python source and inserts Google-style docstring placeholders.
22
23     Args:
24         source_code (str): The raw string content of a .py file.
25
26     Returns:
27         str: The modified source code with docstring templates inserted.
28
29     Example:
30     >>> code = "def add(a, b): return a + b"
31     >>> print(generate_scaffold(code))
32     def add(a, b):
33         """Summary.
34
35         Args:
36             a (type): Description.
37         """
38         return a + b
39     """
40
41     try:
42         tree = ast.parse(source_code)
43     except SyntaxError:
44         return source_code
45
46     lines = source_code.splitlines()
47
48     # Identify Functions and Classes.
49     nodes = [n for n in ast.walk(tree) if isinstance(n, (ast.FunctionDef, ast.ClassDef, ast.AsyncFunctionDef))]
50
51     # We process nodes in reverse order of their line numbers.
52     # This is vital because inserting text shifts the line numbers of everything
53     # below the insertion point. By starting at the bottom, we preserve the
54     # coordinate system for the nodes above.
55     nodes.sort(key=lambda x: x.lineno, reverse=True)
56
57     for node in nodes:
58         # Skip nodes that already have documentation to avoid duplication.
59         if ast.get_docstring(node):
60             continue
61
62         # Use col_offset to determine the exact indentation level.
63         # This ensures the docstring aligns perfectly with the function body.
64         indent = " " * node.col_offset
65         inner_indent = indent + " "
66
67         if isinstance(node, ast.FunctionDef, ast.AsyncFunctionDef):
68             # Filter out 'self' and 'cls' as they typically aren't documented in args.
69             params = [arg.arg for arg in node.args.args if arg.arg not in ("self", "cls")]
70             arg_string = ", ".join(f"{inner_indent}{arg} (type): Description." for p in params)
71
72             doc = f"""{inner_indent}"""Summary of function {node.name}
73             {inner_indent}Args: {arg_string}
74             {inner_indent}Returns: {inner_indent} type: Description.
75             {inner_indent}"""
76
77         elif isinstance(node, ast.ClassDef):
78             doc = f"""{inner_indent}"""Summary of class {node.name}
79             {inner_indent}Attributes: {inner_indent} attr (type): Description.
80             {inner_indent}"""
81
82         # Now insert the docstring, ensuring it sits above the first line of code.
83         # We immediately follow the 'def' or 'class' statement.
84         lines.insert(node.lineno, doc)
85
86     return "\n".join(lines)
87
88 def main(filename):
89     """Main function that reads a file and writes the scaffolded version back to disk.
90
91     Args:
92         filename (str): Path to the .py file to be processed.
93
94     Returns:
95         None
96
97     """
98     with open(filename, 'r', encoding='utf-8') as f:
99         content = f.read()
100
101     scaffolded = generate_scaffold(content)
102
103     with open(filename, 'w', encoding='utf-8') as f:
104         f.write(scaffolded)
105
106     print(f"Successfully added placeholders to {filename}")

```