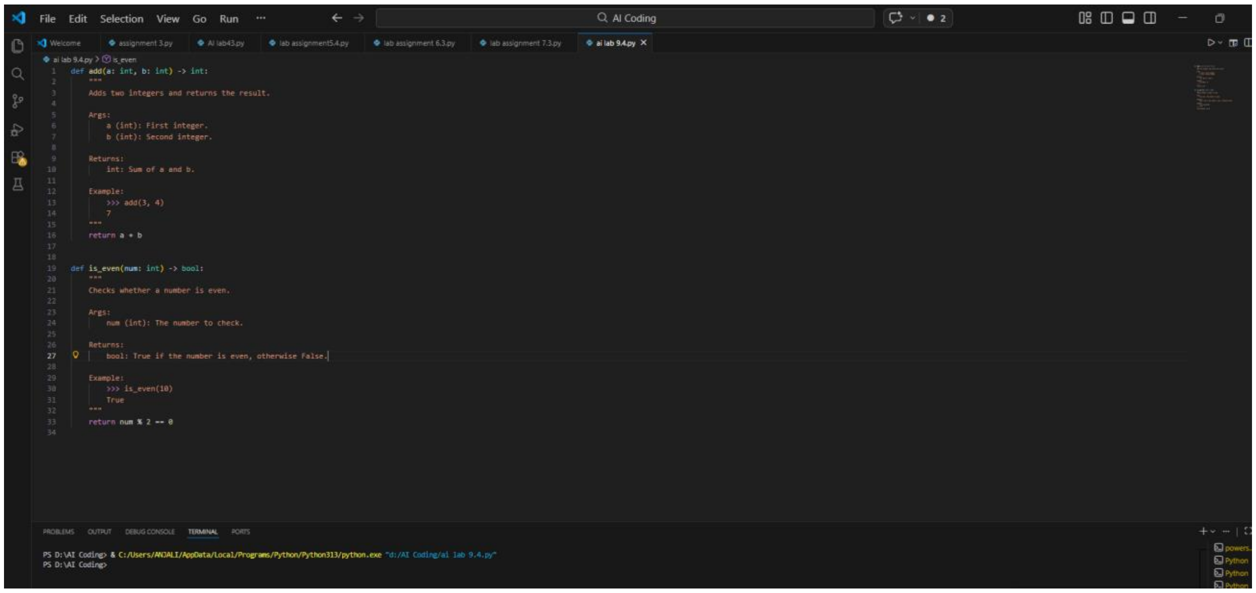
2303a51924

Lab assignment – 9.4

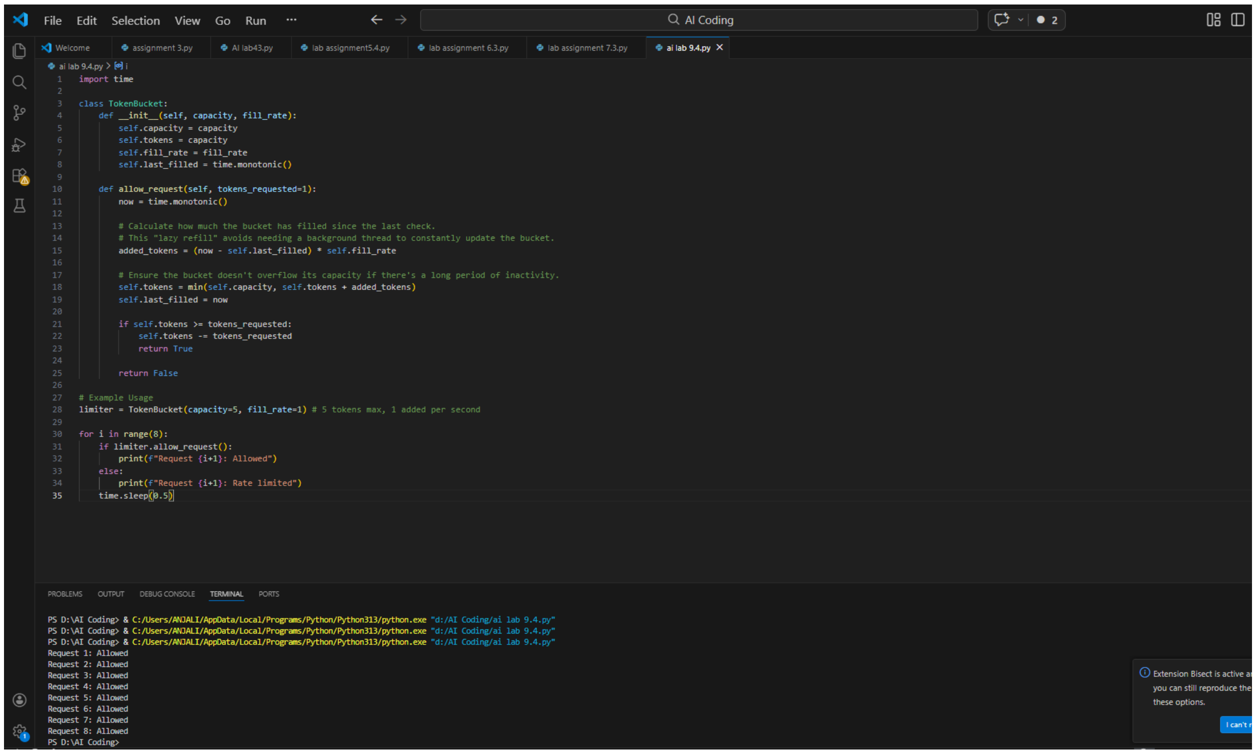
Task 1: Auto-Generating Function Documentation in a Shared Codebase



```python
def add(a: int, b: int) -> int:
    """
    Adds two integers and returns the result.

    Args:
        a (int): First integer.
        b (int): Second integer.

    Returns:
        int: Sum of a and b.

    Example:
        >>> add(3, 4)
        7
    """
    return a + b


def is_even(num: int) -> bool:
    """
    Checks whether a number is even.

    Args:
        num (int): The number to check.

    Returns:
        bool: True if the number is even, otherwise False.

    Example:
        >>> is_even(10)
        True
    """
    return num % 2 == 0
```

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
PS D:\AI Coding>
```

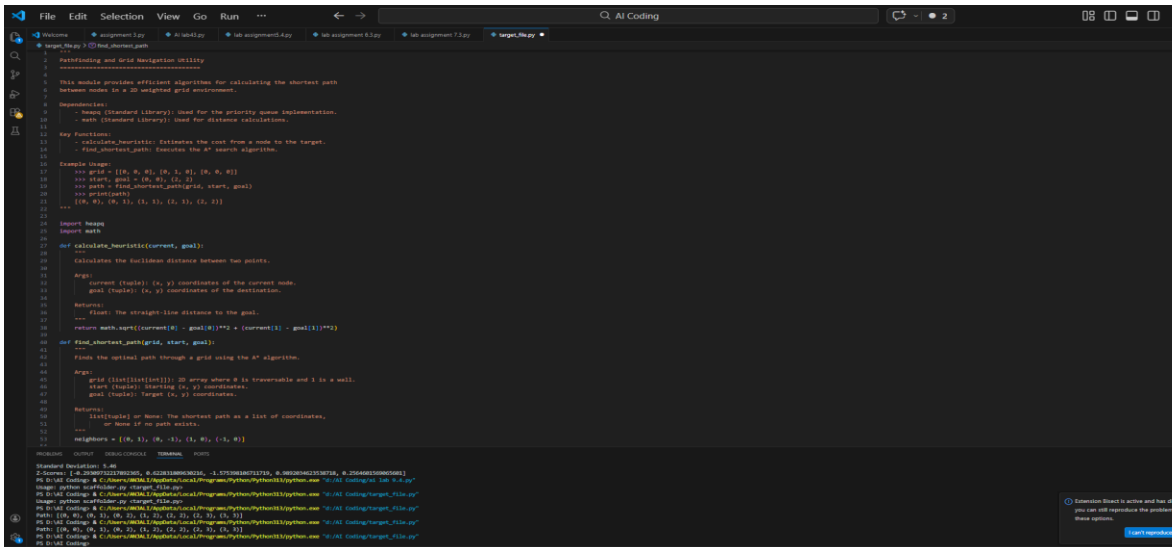
Task 2: Enhancing Readability Through AI-Generated Inline Comments



```python
import time

class TokenBucket:
    def __init__(self, capacity, fill_rate):
        self.capacity = capacity
        self.tokens = capacity
        self.fill_rate = fill_rate
        self.last_filled = time.monotonic()

    def allow_request(self, tokens_requested=1):
        now = time.monotonic()

        # Calculate how much the bucket has filled since the last check.
        # This "lazy refill" avoids needing a background thread to constantly update the bucket.
        added_tokens = (now - self.last_filled) * self.fill_rate

        # Ensure the bucket doesn't overflow its capacity if there's a long period of inactivity.
        self.tokens = min(self.capacity, self.tokens + added_tokens)
        self.last_filled = now

        if self.tokens >= tokens_requested:
            self.tokens -= tokens_requested
            return True

        return False

# Example Usage
limiter = TokenBucket(capacity=5, fill_rate=1) # 5 tokens max, 1 added per second

for i in range(8):
    if limiter.allow_request():
        print(f"Request {i+1}: Allowed")
    else:
        print(f"Request {i+1}: Rate limited")
    time.sleep(0.5)
```

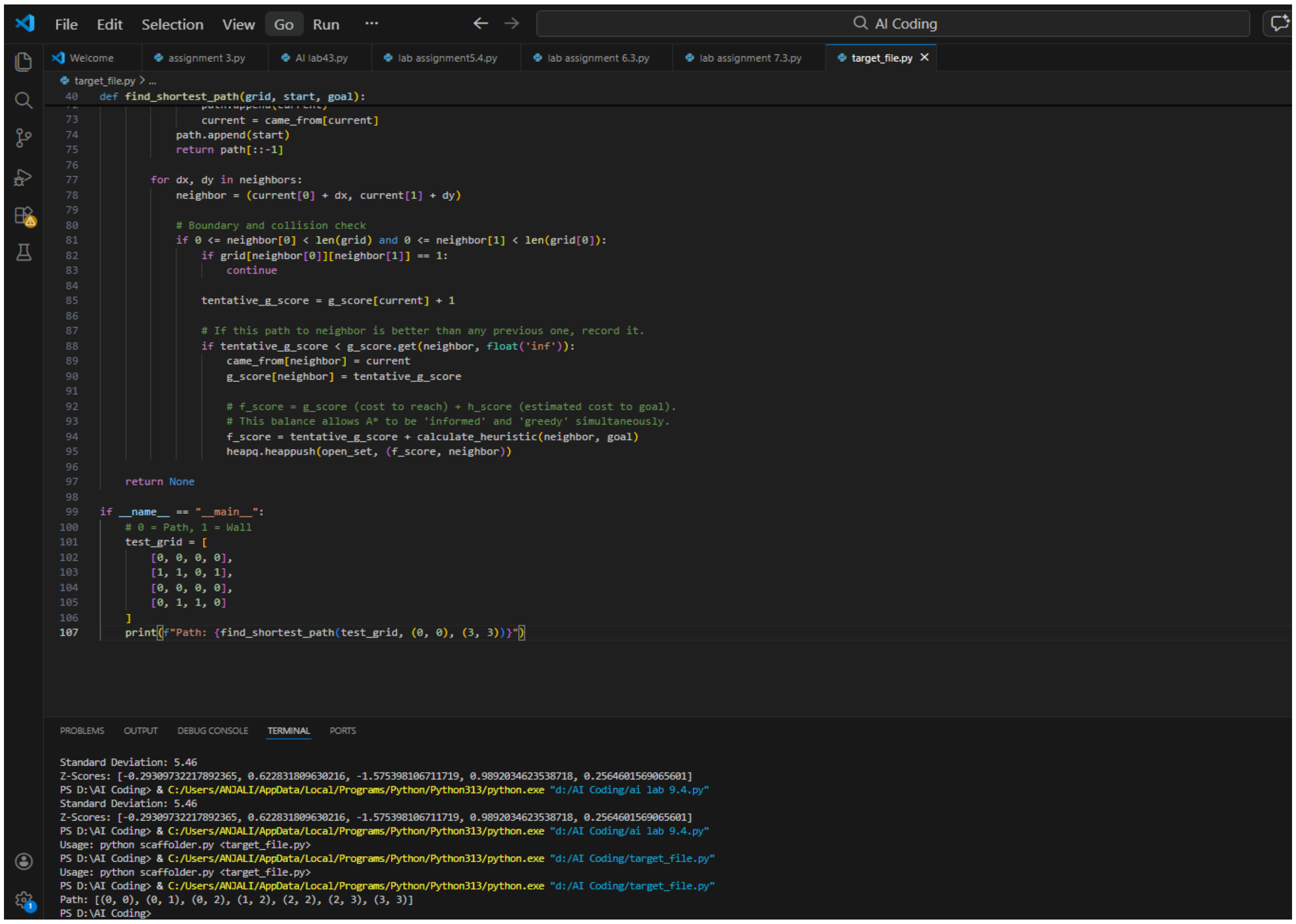
```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "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>
```

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



```python
"""
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, goal = (0, 0), (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)]
```

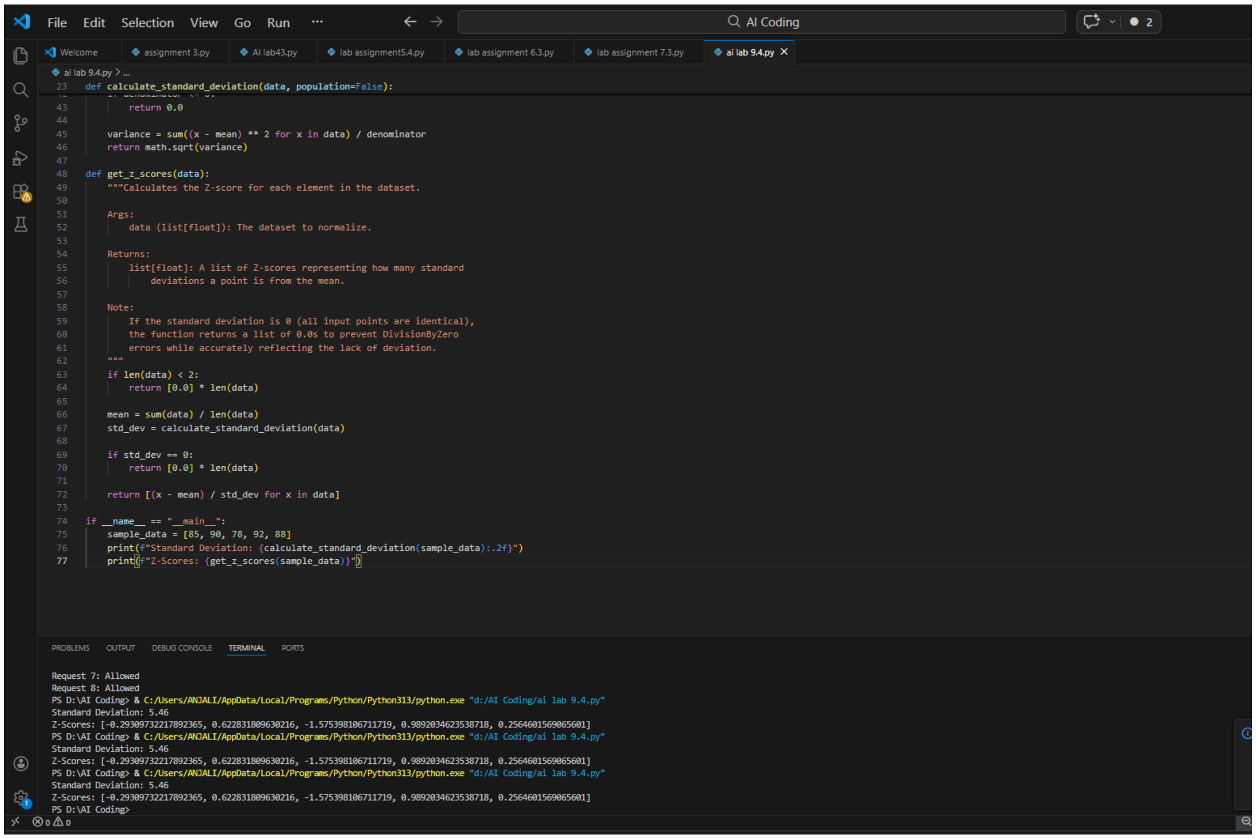
```
Standard Deviation: 5.46
Z-Scores: [-0.29309732217892365, 0.622831809630216, -1.575398106711719, 0.9892034623538718, 0.2564601569065601]
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
Usage: python scaffolder.py <target_file.py>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



```python
def find_shortest_path(grid, start, goal):
                current = came_from[current]
            path.append(start)
            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))}")
```

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

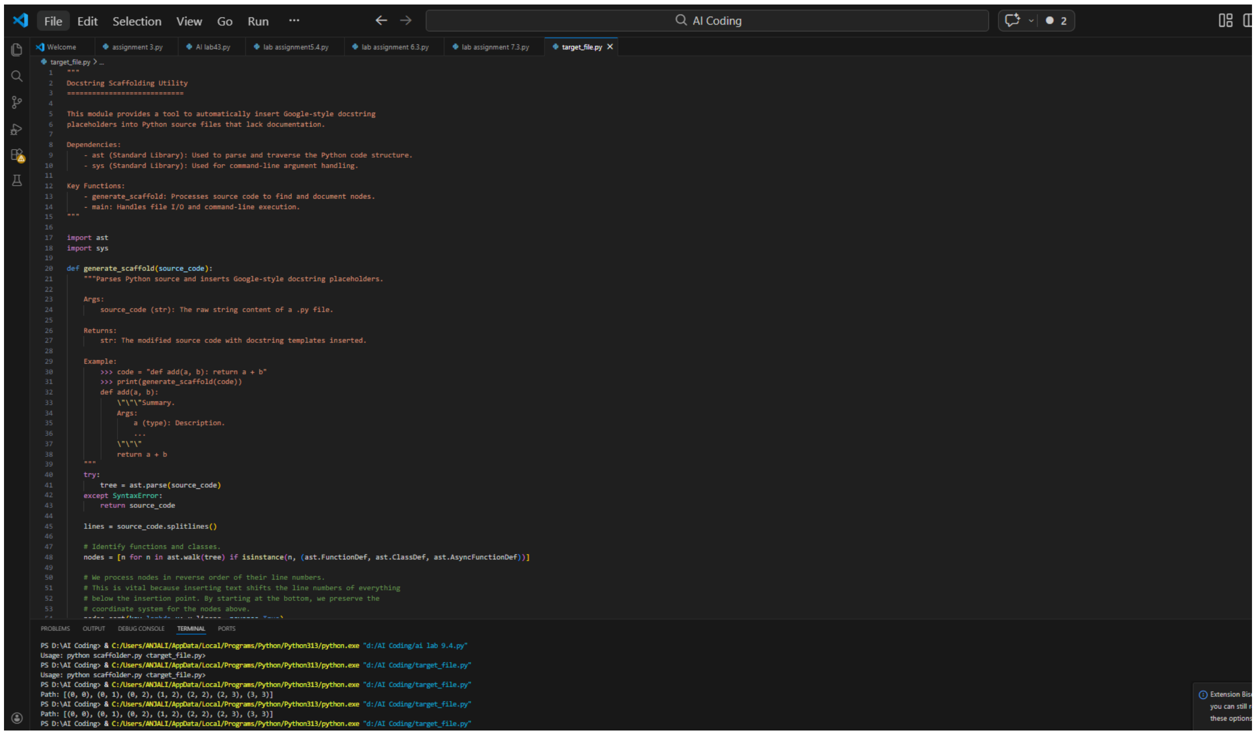


File Edit Selection View Go Run ···
AI Coding
Welcome | assignment 3.py | AI lab43.py | lab assignment5.4.py | lab assignment 6.3.py | lab assignment 7.3.py | ai lab 9.4.py
ai lab 9.4.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

    variance = sum((x - mean) ** 2 for x in data) / denominator
    return math.sqrt(variance)

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)}")
```

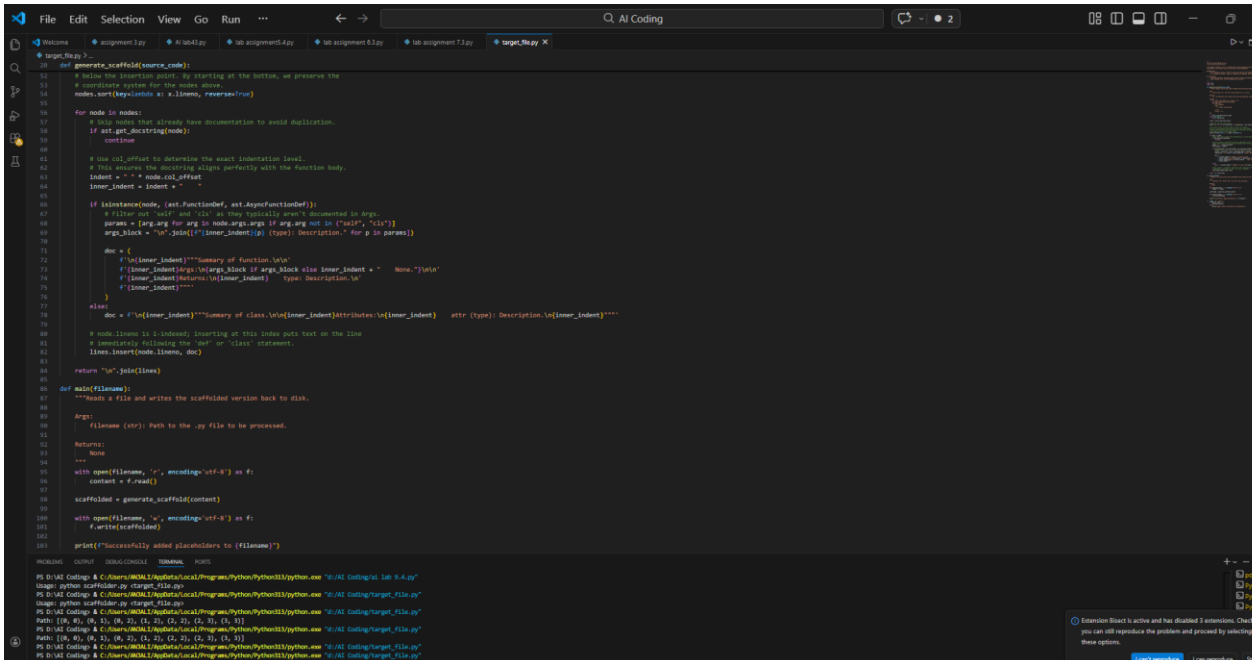
```
Request 7: Allowed
Request 8: Allowed
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
Standard Deviation: 5.46
Z-Scores: [-0.29309732217892365, 0.622831809630216, -1.575398106711719, 0.9892034623538718, 0.2564601569065601]
PS D:\AI Coding> & C:/Users/ANJALI/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/ANJALI/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>
```

Task 5: Building a Mini Automatic Documentation Generator



```python
"""
Docstring Scaffolding Utility
============================

This module provides a tool to automatically insert Google-style docstring
placeholders into Python source files that lack documentation.

Dependencies:
    - ast (Standard Library): Used to parse and traverse the Python code structure.
    - sys (Standard Library): Used for command-line argument handling.

Key Functions:
    - generate_scaffold: Processes source code to find and document nodes.
    - main: Handles file I/O and command-line execution.
"""

import ast
import sys

def generate_scaffold(source_code):
    """Parses Python source and inserts Google-style docstring placeholders.

    Args:
        source_code (str): The raw string content of a .py file.

    Returns:
        str: The modified source code with docstring templates inserted.

    Example:
        >>> code = "def add(a, b): return a + b"
        >>> print(generate_scaffold(code))
        def add(a, b):
            \"\"\"Summary.
            Args:
                a (type): Description.
                ...
            \"\"\"
            return a + b
    """
    try:
        tree = ast.parse(source_code)
    except SyntaxError:
        return source_code

    lines = source_code.splitlines()

    # Identify functions and classes.
    nodes = [n for n in ast.walk(tree) if isinstance(n, (ast.FunctionDef, ast.ClassDef, ast.AsyncFunctionDef))]

    # We process nodes in reverse order of their line numbers.
    # This is vital because inserting text shifts the line numbers of everything
    # below the insertion point. By starting at the bottom, we preserve the
    # coordinate system for the nodes above.
```

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
Usage: python scaffolder.py <target_file.py>
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
Usage: python scaffolder.py <target_file.py>
PS D:\AI Coding> & C:/Users/ANJALI/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/ANJALI/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/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
```



```python
def generate_scaffold(source_code):
    # below the insertion point. By starting at the bottom, we preserve the
    # coordinate system for the nodes above.
    nodes.sort(key=lambda x: x.lineno, reverse=True)

    for node in nodes:
        # Skip nodes that already have documentation to avoid duplication.
        if ast.get_docstring(node):
            continue

        # Use col_offset to determine the exact indentation level.
        # This ensures the docstring aligns perfectly with the function body.
        indent = " " * node.col_offset
        inner_indent = indent + "    "

        if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef)):
            # Filter out 'self' and 'cls' as they typically aren't documented in Args.
            params = [arg.arg for arg in node.args.args if arg.arg not in ("self", "cls")]
            args_block = "\n".join([f"{inner_indent}{p} (type): Description." for p in params])

            doc = (
                f'\n{inner_indent}"""Summary of function.\n\n'
                f'{inner_indent}Args:\n{args_block if args_block else inner_indent + "    None."}\n\n'
                f'{inner_indent}Returns:\n{inner_indent}    type: Description.\n'
                f'{inner_indent}"""'
            )
        else:
            doc = f'\n{inner_indent}"""Summary of class.\n\n{inner_indent}Attributes:\n{inner_indent}    attr (type): Description.\n{inner_indent}"""'

        # node.lineno is 1-indexed; inserting at this index puts text on the line
        # immediately following the 'def' or 'class' statement.
        lines.insert(node.lineno, doc)

    return "\n".join(lines)

def main(filename):
    """Reads a file and writes the scaffolded version back to disk.

    Args:
        filename (str): Path to the .py file to be processed.

    Returns:
        None
    """
    with open(filename, 'r', encoding='utf-8') as f:
        content = f.read()

    scaffolded = generate_scaffold(content)

    with open(filename, 'w', encoding='utf-8') as f:
        f.write(scaffolded)

    print(f"Successfully added placeholders to {filename}")
```

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/ai lab 9.4.py"
Usage: python scaffolder.py <target_file.py>
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
Usage: python scaffolder.py <target_file.py>
PS D:\AI Coding> & C:/Users/ANJALI/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/ANJALI/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/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/target_file.py"
```