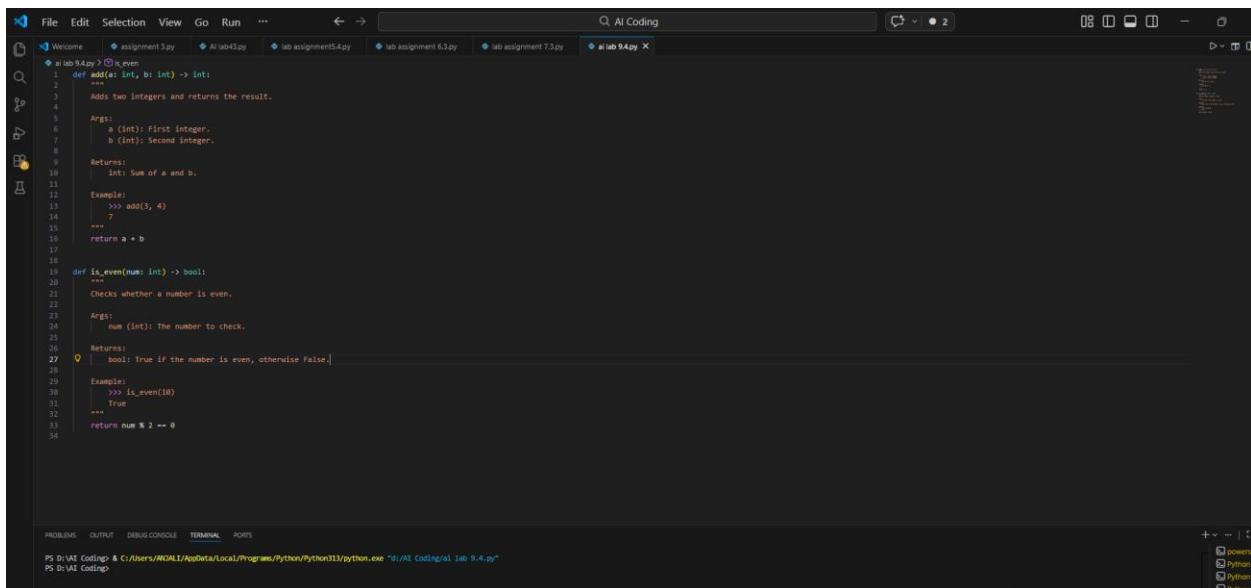


2303a51924

Lab assignment – 9.4

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



The screenshot shows a code editor window with a dark theme. The file being edited is `lab 9.4.py`. The code contains two functions: `add` and `is_even`, each with detailed docstrings. The `add` function adds two integers and returns the result. The `is_even` function checks whether a number is even. The code editor interface includes a top navigation bar with File, Edit, Selection, View, Go, Run, etc., and a bottom toolbar with PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS.

```
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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\AI Coding & C:\Users\WDALE\Appldata\Local\Programs\Python\Python311\python.exe "d:\AI Coding\lab 9.4.py"

PS D:\AI Coding

Task 2: Enhancing Readability Through AI-Generated Inline Comments

The screenshot shows a Python file named `ai lab 9.4.py` in the VS Code editor. The code defines a `TokenBucket` class with methods for initializing capacity and fill rate, calculating tokens available, and allowing requests. The code is annotated with numerous inline comments explaining its logic and performance optimizations like "lazy refill". The terminal below shows the script running and accepting 8 requests, all of which are allowed.

```
File Edit Selection View Go Run ... ⏪ ⏩ 🔍 AI Coding
ai lab 9.4.py | i
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(f"Request {i+1}: Allowed")
33     else:
34         print(f"Request {i+1}: Rate limited")
35     time.sleep(0.5)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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
PS D:\AI Coding
```

Extension Bisect is active and you can still reproduce the these options.
I can't find it

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



The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, ...
- Toolbar:** Welcome, assignment1.py, AIWorld.py, lab assignment 4.py, lab assignment 6.5.py, lab assignment 7.3.py, target_file.py
- Code Editor:** The main window displays a Python script for the A* search algorithm. The code includes imports for heapq and math, defines calculate_heuristic and find_shortest_path functions, and provides example usage with a grid and start/goal coordinates.
- Bottom Status Bar:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS.
- Bottom Log:** Standard Deviation: 5.46. A series of log entries from 'D:\UVic Coding' showing file operations like opening and saving files.
- Bottom Right Corner:** A tooltip from the Extension Toolkit plugin: "Extension Toolkit is active and ready. You can still reproduce the problem. These options..."

The screenshot shows a code editor window titled "AI Coding". The top bar includes standard menu items like File, Edit, Selection, View, Go, Run, and a Help icon. Below the menu is a search bar labeled "AI Coding". The main area displays Python code for finding the shortest path in a grid using A* search. The code defines a function `find_shortest_path` that takes a grid, start coordinates, and goal coordinates. It uses a priority queue `open_set` to manage nodes based on their f-score (g-score + heuristic). The g-score is updated if a better path is found. The heuristic is calculated as the Manhattan distance from the current node to the goal. A test grid is provided at the end of the function. The bottom of the screen shows a standard terminal window with command-line output related to the execution of the script.

```
File Edit Selection View Go Run ...
Welcome assignment 3.py AI lab43.py lab assignment54.py lab assignment 6.3.py lab assignment 7.3.py target_file.py

target_file.py > ...
40 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: [-9.2930972217892365, 0.622831890630216, -1.575398106711719, 0.9892034623538718, 0.2564601569065601]
PS D:\AI Coding & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/ai lab 5.4.py"
Standard Deviation: 5.46
Z-Scores: [-9.2930972217892365, 0.622831890630216, -1.575398106711719, 0.9892034623538718, 0.2564601569065601]
PS D:\AI Coding & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/ai lab 5.4.py"
Usage: python scarfolder.py <target_file.py>
PS D:\AI Coding & C:\Users\MDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/target_file.py"
Usage: python scarfolder.py <target_file.py>
PS D:\AI Coding & C:\Users\MDALI\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 developer comments within the code have been converted into structured docstrings using triple quotes. The `calculate_standard_deviation` function includes notes about handling zero standard deviation and provides a detailed note about the function's behavior when all input points are identical. The `get_z_scores` function includes an argument description for `data` and a return value description for the list of Z-scores.

```
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 the resulting 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, 0.622831889630216, -1.575398106711719, 0.9892934623538719, 0.2564601569065691]
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, 0.622831889630216, -1.575398106711719, 0.9892934623538719, 0.2564601569065691]
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, 0.622831889630216, -1.575398106711719, 0.9892934623538719, 0.2564601569065691]
PS D:\AI Coding
```

Task 5: Building a Mini Automatic Documentation Generator

```

File Edit Selection View Go Run ...
File Edit Selection View Go Run ...
Welcome target_file.py AI lab3.py AI lab4.py AI assignment5.py AI assignment6.py AI assignment7.py target_file.py
1 Docstring Scaffolding Utility
2 =====
3 This module provides a tool to automatically insert Google-style docstring
4 placeholders into Python source files that lack documentation.
5
6 Dependencies:
7   - ast (Standard Library): Used to parse and traverse the Python code structure.
8   - sys (Standard Library): Used for command-line argument handling.
9
10 Key Functions:
11   generate_scaffold: Processes source code to find and document nodes.
12   main: Handles file I/O and command-line execution.
13
14 import ast
15 import sys
16
17 def generate_scaffold(source_code):
18     """Parses Python source and inserts Google-style docstring placeholders.
19
20     Args:
21         source_code (str): The raw string content of a .py file.
22
23     Returns:
24         str: The modified source code with docstring templates inserted.
25
26     Example:
27         >>> code = '<def add(a, b): return a + b>' 
28         >>> print(generate_scaffold(code))
29         def add(a, b):
30             """Summary.
31             Args:
32                 a (type): Description.
33
34             Returns:
35                 b (type): Description.
36
37             >>> return a + b
38
39     try:
40         tree = ast.parse(source_code)
41     except SyntaxError:
42         return source_code
43
44     lines = source_code.splitlines()
45
46     # Identify functions and classes.
47     nodes = [n for n in ast.walk(tree) if isinstance(n, (ast.FunctionDef, ast.ClassDef, ast.AsyncFunctionDef))]
48
49     # We process nodes in reverse order of their line numbers.
50     # This is vital because inserting text shifts the line numbers of everything
51     # below the insertion point. By starting at the bottom, we preserve the
52     # coordinate system for the nodes above.
53
54     for node in nodes:
55         # Skip nodes that already have documentation to avoid duplication.
56         if ast.get_docstring(node):
57             continue
58
59         # Use col_offset to determine the exact indentation level.
60         # This ensures the docstring aligns perfectly with the function body.
61         indent = '    ' * node.col_offset
62         inner_indent = indent + '    '
63
64         if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef)):
65             # Function or class definition.
66             args = node.args
67             arg_descs = [arg.arg for arg in args.args if arg.arg not in ('self', 'cls')]
68             args_block = '\n'.join(['        """{}: {}"'.format(arg_desc, type(arg_desc)) for arg_desc in arg_descs])
69
70             doc = (
71                 '#' * len(inner_indent)) + 'Summary of function.' + '\n'
72             doc += inner_indent + args_block + inner_indent + '    None.' + '\n'
73             doc += inner_indent + '    Type description.' + '\n'
74             doc += '#' * len(inner_indent)
75
76         else:
77             doc = '#' * len(inner_indent) + 'Summary of class.' + '\n' + inner_indent + attr(type): description.\n' + inner_indent + '    '
78
79         # node.lineno is 1-indexed; inserting at this index puts text on the line
80         # immediately following the 'def' or 'class' statement.
81         lines.insert(node.lineno, doc)
82
83     return '\n'.join(lines)
84
85 def main(filename):
86     """Reads a file and writes the scaffolded version back to disk.
87
88     Args:
89         filename (str): Path to the .py file to be processed.
90
91     Returns:
92         None
93
94     With open(filename, 'r', encoding='utf-8') as f:
95         content = f.read()
96
97         scaffolded = generate_scaffold(content)
98
99         with open(filename, 'w', encoding='utf-8') as f:
100             f.write(scaffolded)
101
102     print("Successfully added placeholders to {}".format(filename))
103
104
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\AI Coding & C:\Users\MOUL\Apidata\local\Programs\Python\Python311\python.exe "D:\AI Coding\ai_lab_9.4.py"
Usage: python scaffold.py target_file.py
PS D:\AI Coding & C:\Users\MOUL\Apidata\local\Programs\Python\Python311\python.exe "D:\AI Coding\target_file.py"
PS D:\AI Coding & C:\Users\MOUL\Apidata\local\Programs\Python\Python311\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\MOUL\Apidata\local\Programs\Python\Python311\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\MOUL\Apidata\local\Programs\Python\Python311\python.exe "D:\AI Coding\target_file.py"
Extension Block is active and has disabled 3 extensions. One you can still reproduce the problem and proceed by selecting these options.

```