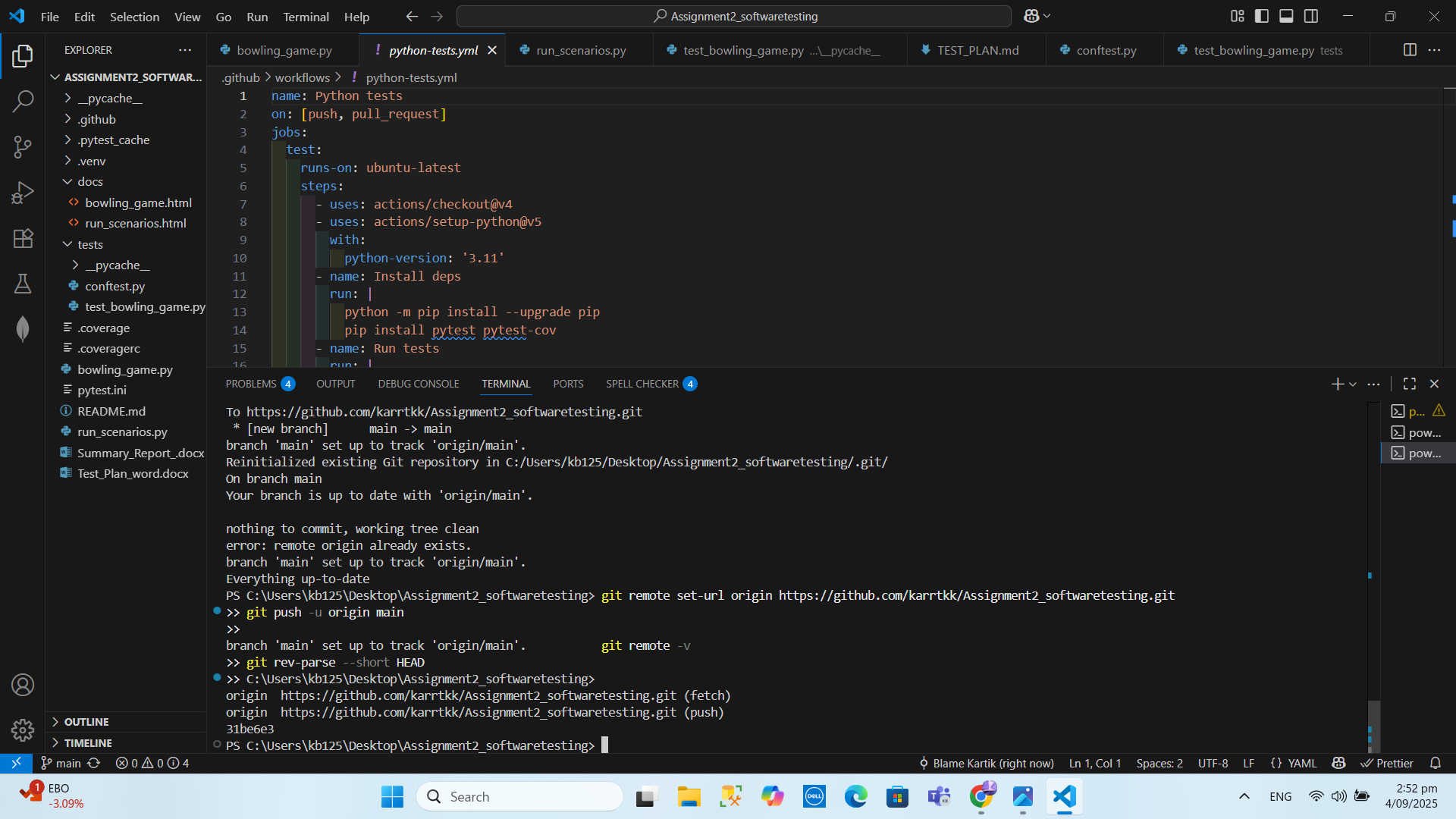
Summary Report – Bowling Game

Name: Kartik Bansal

# GIT HUB: -

Repository: <https://github.com/karrtkk/Assignment2_softwaretesting>



# 1) Executive Summary

A ten-pin bowling scorer was implemented and verified with automated unit tests. The suite covers all scoring rules (strike, spare, open frame, and 10th-frame bonuses) and input validation. All tests passed, and coverage reached 100% lines in the final run. Screenshots are embedded once each in the relevant sections.

# 2) System Under Test

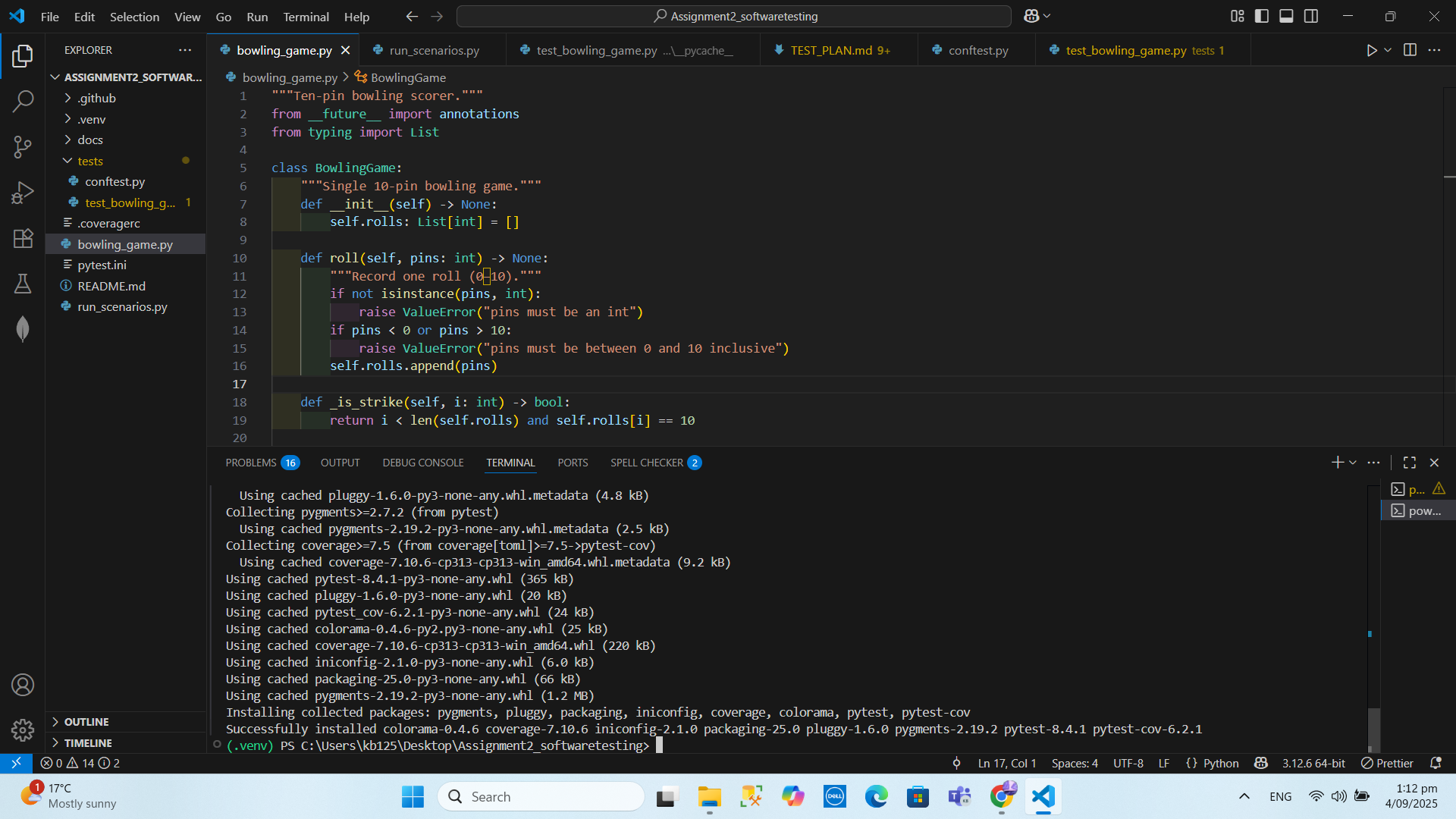
Class: BowlingGame (bowling\_game.py)  
• roll (pins: int) — records one roll (0–10) with validation  
• score() — computes the total for 10 frames, applying spare/strike bonuses and 10th-frame rules

# 3) Test Approach

Tests were designed directly from the bowling rules (black-box) and reviewed to ensure branch coverage for strike, spare, and open paths (white-box). We used pytest for automation and pytest-cov to measure coverage. A small scenario runner (run\_scenarios.py) gives human-readable results.

# 4) Environment & Setup (Evidence)

Python virtual environment and tools installed:

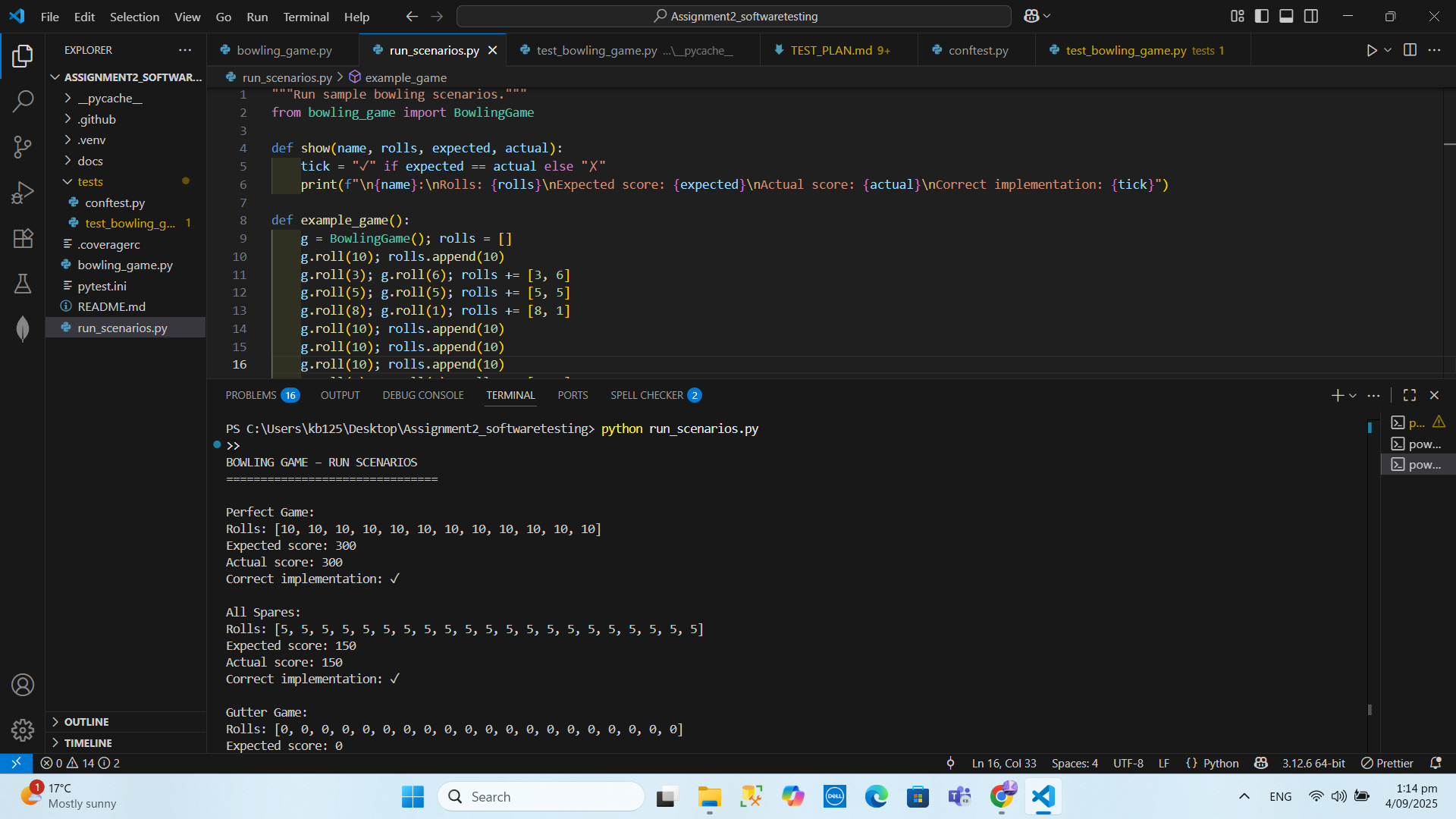


Virtual environment active and pytest/coverage installed in VS Code terminal.

# 5) Unit Test Design & Results

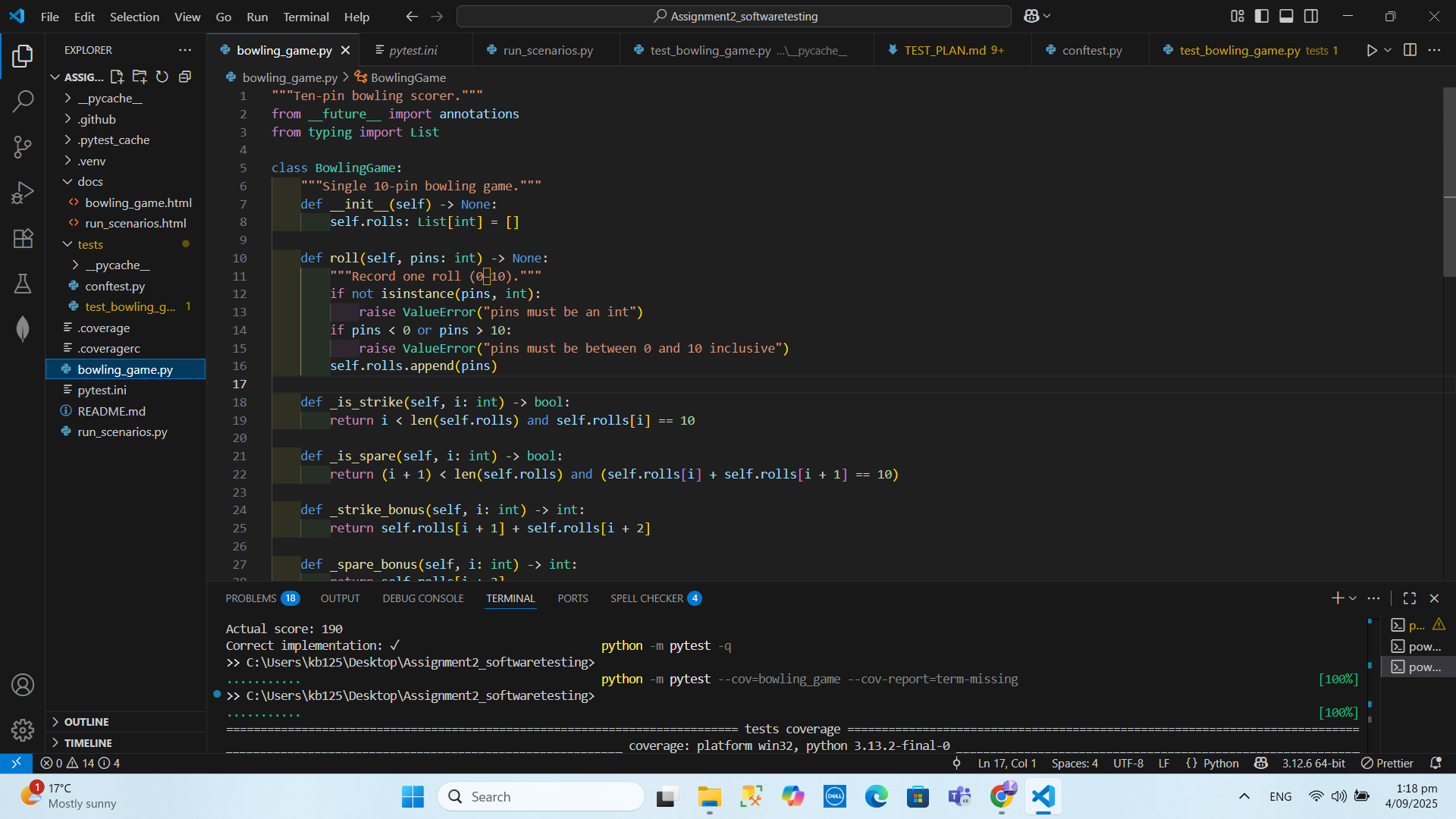
Covered cases:  
• Gutter game (all zeros)  
• All ones  
• One spare followed by a 3  
• One strike followed by 3 and 4  
• All spares → 150  
• Perfect game → 300  
• Regular no-marks game → 72  
• Mixed example from brief → 190

Scenario runner output:



Output of run\_scenarios.py showing scenario totals and checks.

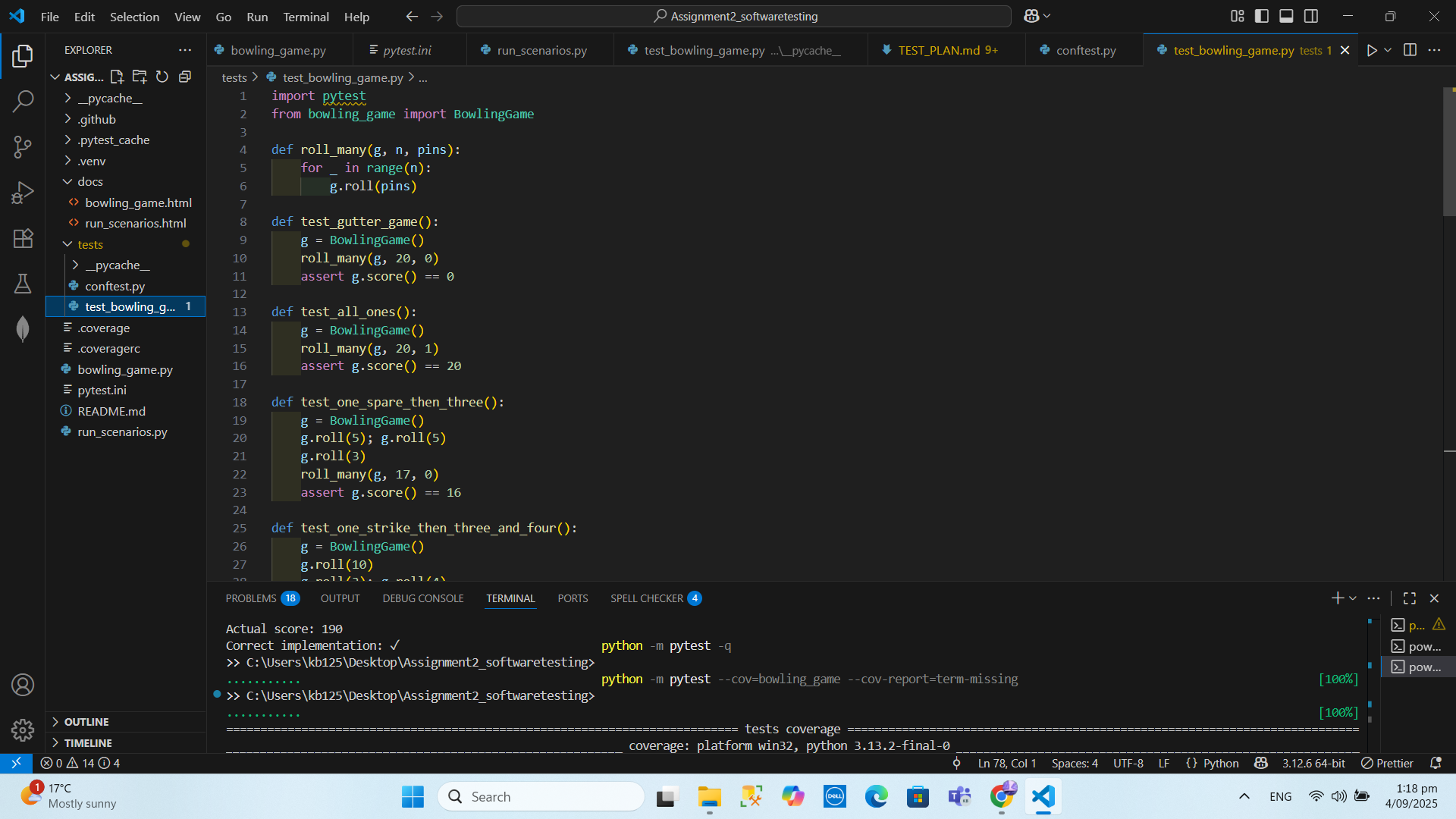
pytest results:



pytest run with all tests passing.

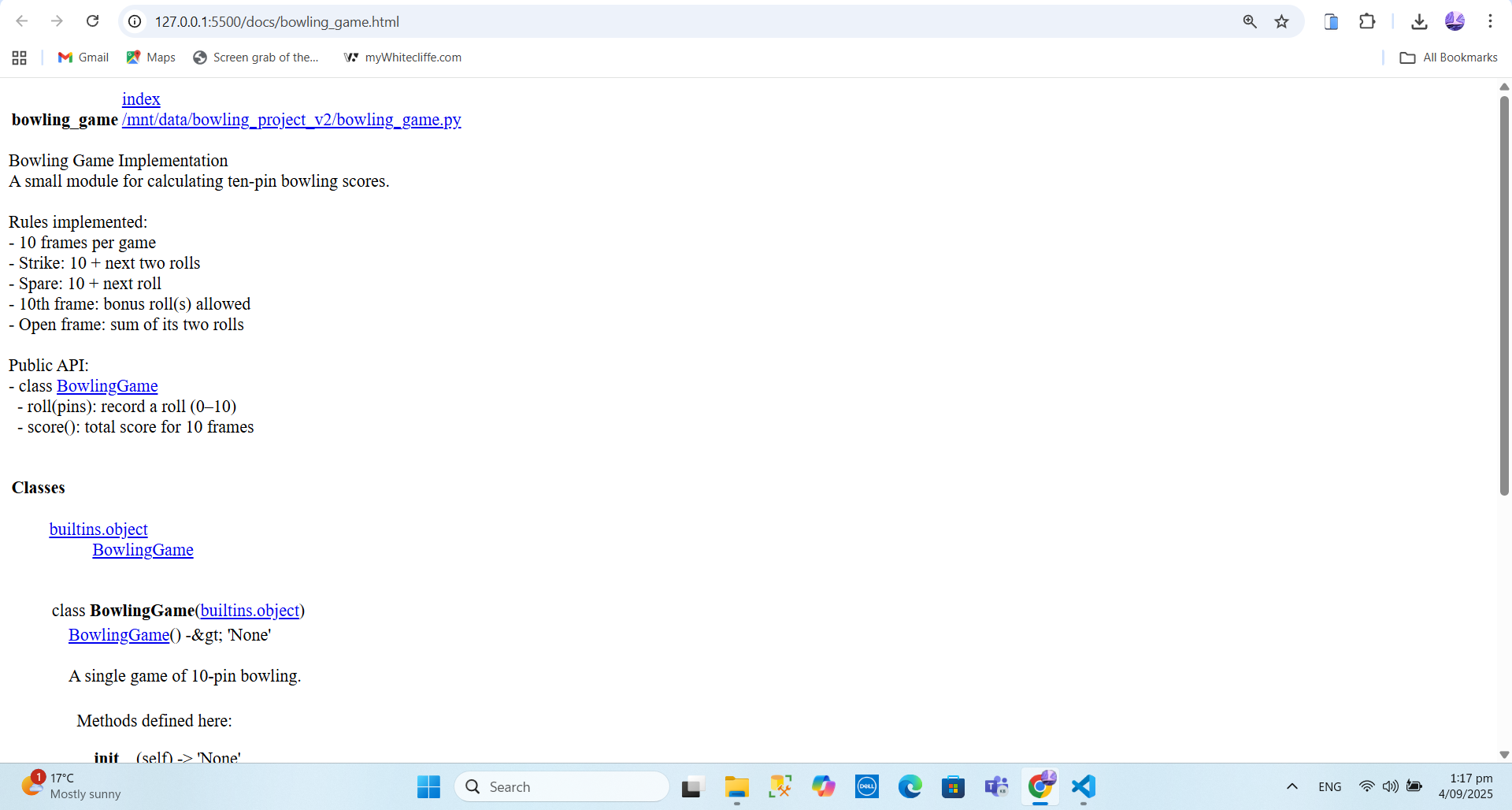
# 6) Coverage

Coverage result for bowling\_game.py:

Coverage table for bowling\_game.py with 100% lines.

# 7) Documentation

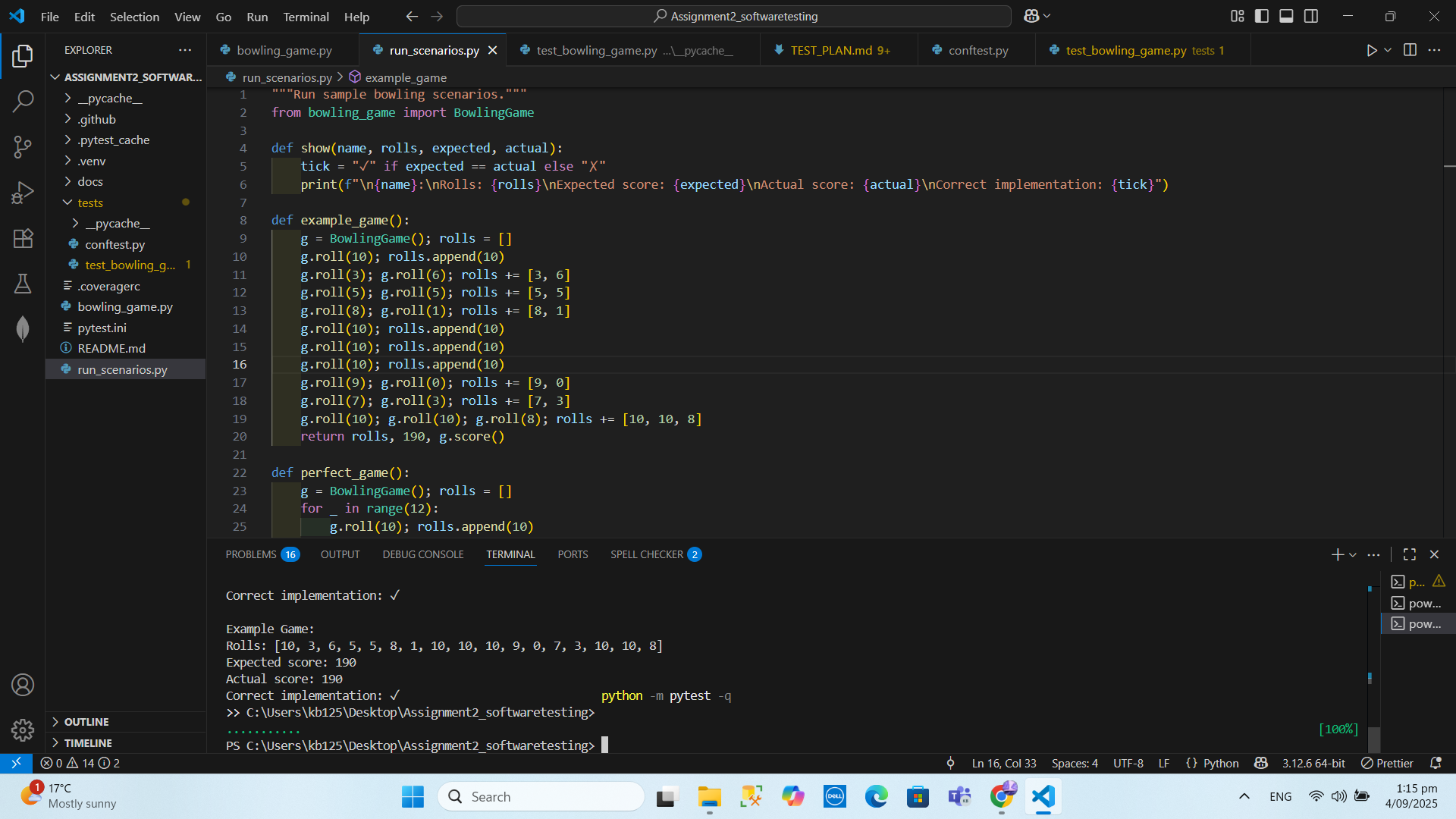
Generated API docs (HTML) for the module:



API docs page header (docs/bowling\_game.html) opened in browser.

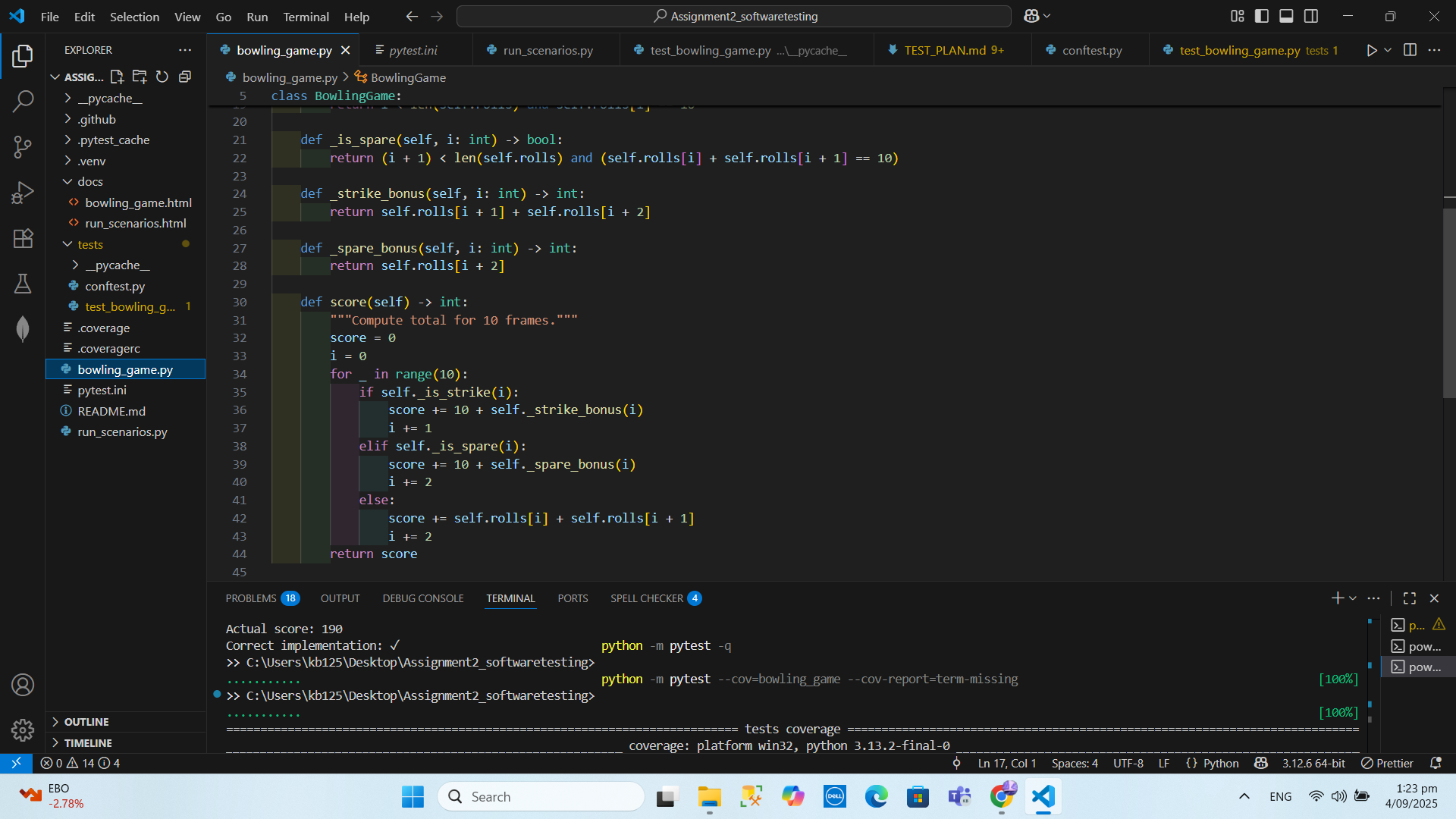
# 8) Implementation: Key Code Views

Class top (\_\_init\_\_ and roll):



Top of bowling\_game.py showing class, \_\_init\_\_, and roll().

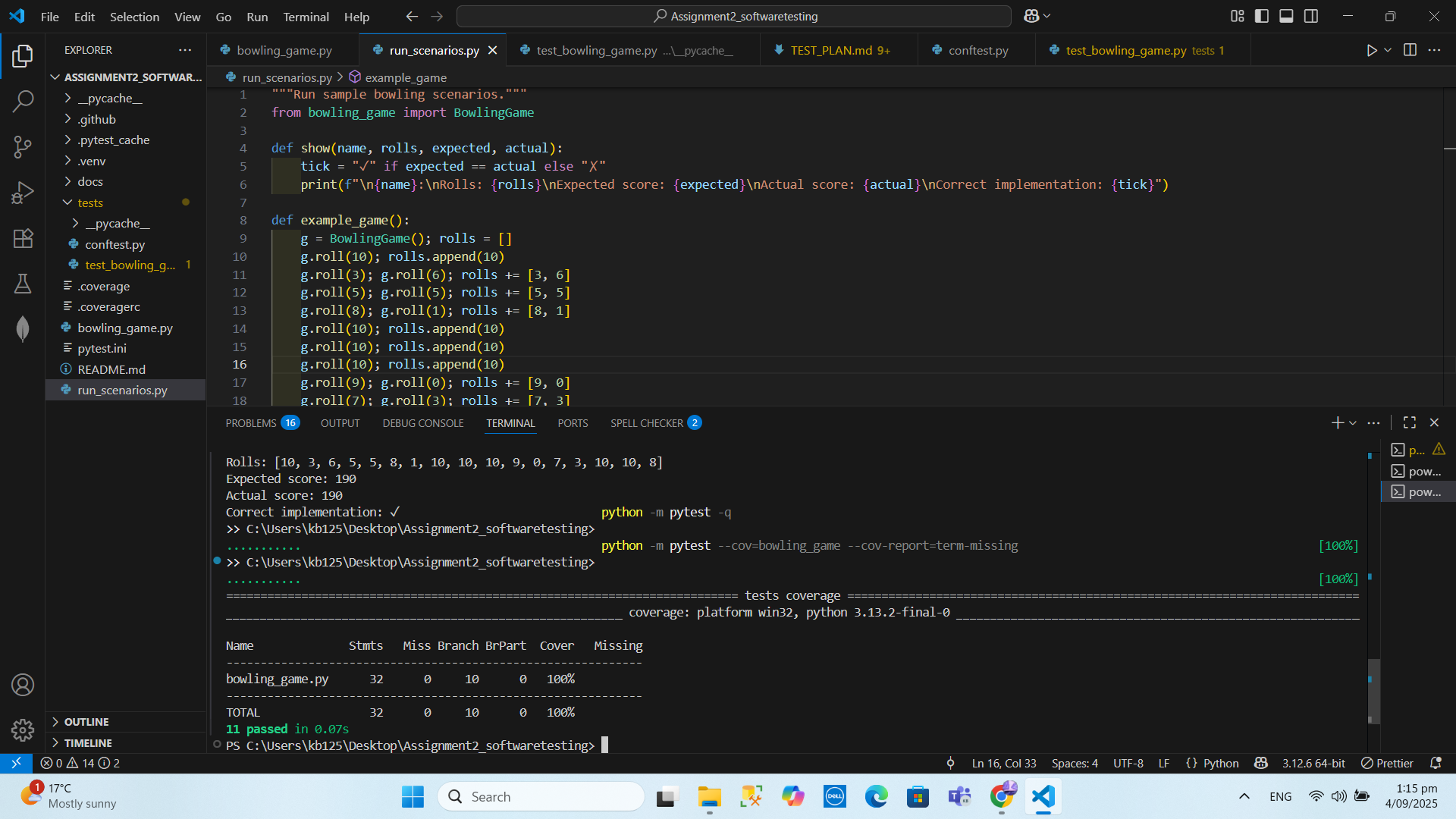
Full score() function:



Full score() function in bowling\_game.py.

# 9) Tests File View

Unit tests file open in VS Code:



tests/test\_bowling\_game.py open in VS Code with multiple tests visible.

# 10) Debugging & Fixes

• Evaluated exactly 10 frames in the scoring loop  
• Open frames sum both rolls  
• Strike/spare bonuses use correct look-ahead values  
• roll() validates input (type int, 0 ≤ pins ≤ 10)

# 11) Refactoring

Refactoring introduced small helpers (\_is\_strike, \_is\_spare, \_strike\_bonus, \_spare\_bonus) and type hints, improving readability and maintainability without changing behavior.

# 12) Learning Outcomes

LO1: QA through unit tests and coverage.  
LO2: Planned, designed, executed, and managed testing activities with pytest.  
LO4: Applied maintenance (bug fixes, refactoring) to improve quality.  
LO5: Version control and CI guidance included.  
LO6: Produced clear developer/user documentation (pydoc + this report).

# 13) Rubric Alignment

Test Planning: Test Plan document.  
Unit Test Design: This section and tests/test\_bowling\_game.py.  
Debugging Implementation: Fixes section and scenario evidence.  
Code Refactoring: Helpers and structure documented above.  
Version Control: Commit strategy and CI workflow described.  
Documentation: pydoc evidence and this report.  
Summary Report: This document consolidates the results.

# 14) Results & Recommendations

All tests passed; final coverage reached 100%. Scenario totals matched expected values. Recommended next steps: add a scoreboard and multiplayer sequencing, integrate a GUI and input layer, and keep CI with a ≥95% coverage gate.