# Question Paper Set A1

AI-Assisted Coding Exam — Python (1 hour)

* You may either (a) write the exact prompt to have your AI assistant generate code, or (b) write the code yourself. If you use AI, submit the prompt and the final code you executed.
* Use VS Code Copilot / Gemini / CursorAI. Credit your prompts.
* Prefer Python standard library; write clear, tested, well-documented code.
* Each subgroup (A–O) has two tasks. Attempt all. Medium complexity, use-case based.
* Syllabus pillars:

A–C: AI completion (classes, loops, conditionals) D–E: Debugging with AI

F–G: Code Quality & Performance

H–I: Documentation & Code Review J–K: Code Quality & Performance L–M: Debugging with AI

N–O: AI completion (classes, loops, conditionals)

Deliverables for each question

1. If using AI: the exact prompt you issued. If manual: note 'manual' and a brief design reason.
2. solution.py
3. tests.py (unittest/pytest; write tests first for TDD items)
4. Docstrings & inline comments (AI-assisted allowed)
5. Short README.md (approach, assumptions, complexity, run tests)
6. For debugging/refactor: brief before/after note

Question Paper Set A1

AI-Assisted Coding Exam — Python (1 hour)

* + You may either (a) write the exact prompt to have your AI assistant generate code, or (b) write the code yourself. If you use AI, submit the prompt and the final code you executed.
  + Use VS Code Copilot / Gemini / CursorAI. Credit your prompts.
  + Prefer Python standard library; write clear, tested, well-documented code.
  + Each subgroup (A–O) has two tasks. Attempt all. Medium complexity, use-case based.
  + Syllabus pillars:

A–C: AI completion (classes, loops, conditionals) D–E: Debugging with AI

F–G: TDD with AI

H–I: Documentation & Code Review J–K: Code Quality & Performance L–M: Files/CSV & Regex

N–O: AI completion (classes, loops, conditionals)

Deliverables for each question

1. If using AI: the exact prompt you issued. If manual: note 'manual' and a brief design reason.
2. solution.py
3. tests.py (unittest/pytest; write tests first for TDD items)
4. Docstrings & inline comments (AI-assisted allowed)
5. Short README.md (approach, assumptions, complexity, run tests)
6. For debugging/refactor: brief before/after note

# Subgroup F — AI-Assisted Code Review

* 1. **— [S01F1] Inefficient Prime Checker**

## Scenario (Code Review):

Context: A student wrote code to check if a number is prime, but it’s inefficient.

## Buggy Code:

def is\_prime(n): if n <= 1:

return False

for i in range(2, n): if n % i == 0:

return False return True

## Task:

* Use AI-assisted code review to suggest improvements.
* Optimize loop range, variable naming, and add docstring.

## Sample Input/Output:

Input: 29 → Output: True Input: 12 → Output: False

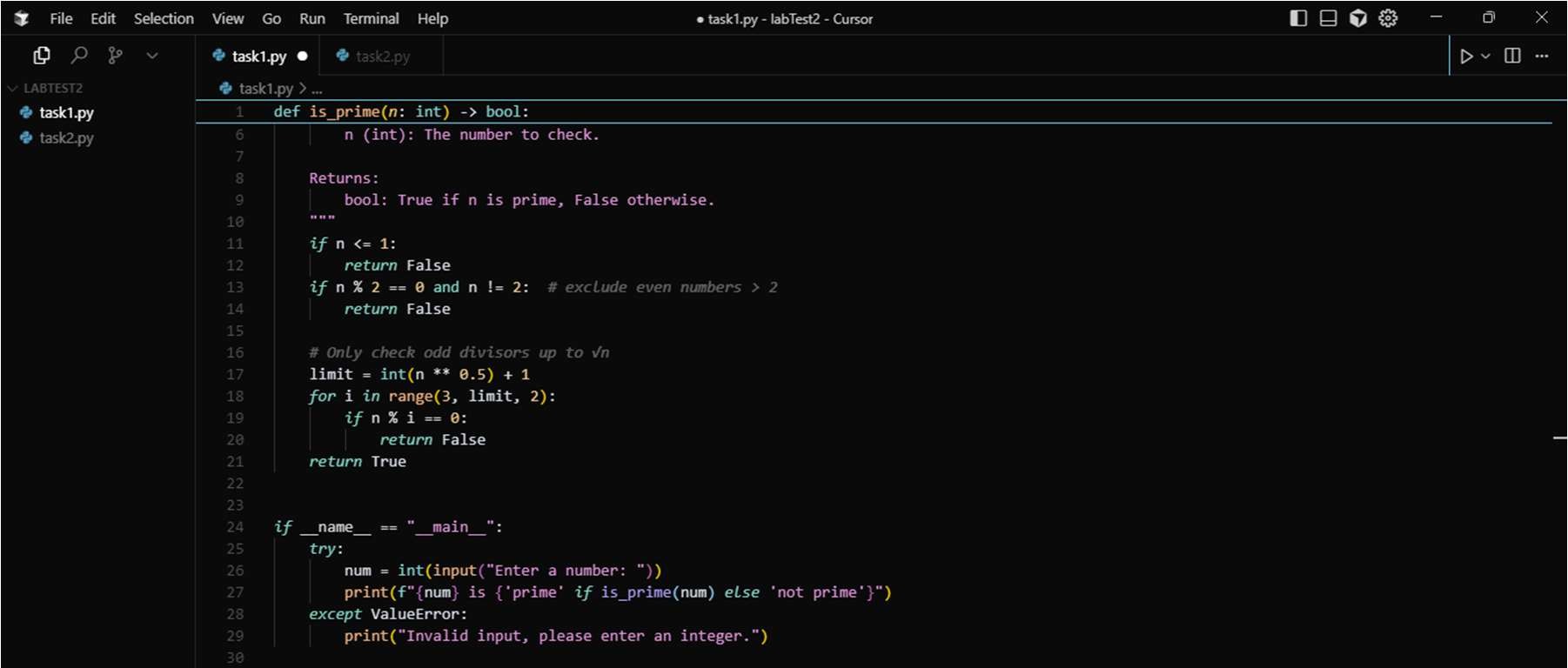
## Acceptance Criteria:

* Efficient up to √n.
* Clear variable names.
* Includes docstring.

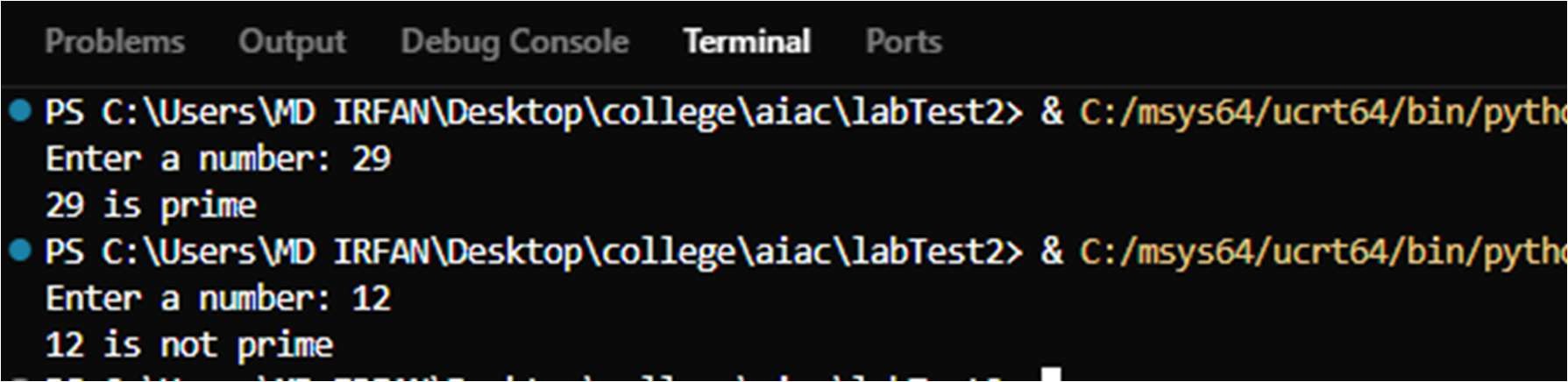
Solution:

PROMPT : Review and optimize this inefficient prime-checking function. Improve loop efficiency using √n, rename variables for clarity, and add a proper docstring and Totally debug this code.

CORRECTED CODE:



OUTPUT:



* 1. **— [S01F2] Readability in Student Grade Script**

## Scenario:

The following script calculates averages but is unreadable and unmaintainable.

## Buggy Code:

def f(l):

s = 0

for i in l:

s += i return s/len(l)

## Task:

* Use AI to review code and refactor for readability.
* Add type hints, descriptive names, error handling for empty list.

## Expected Output:

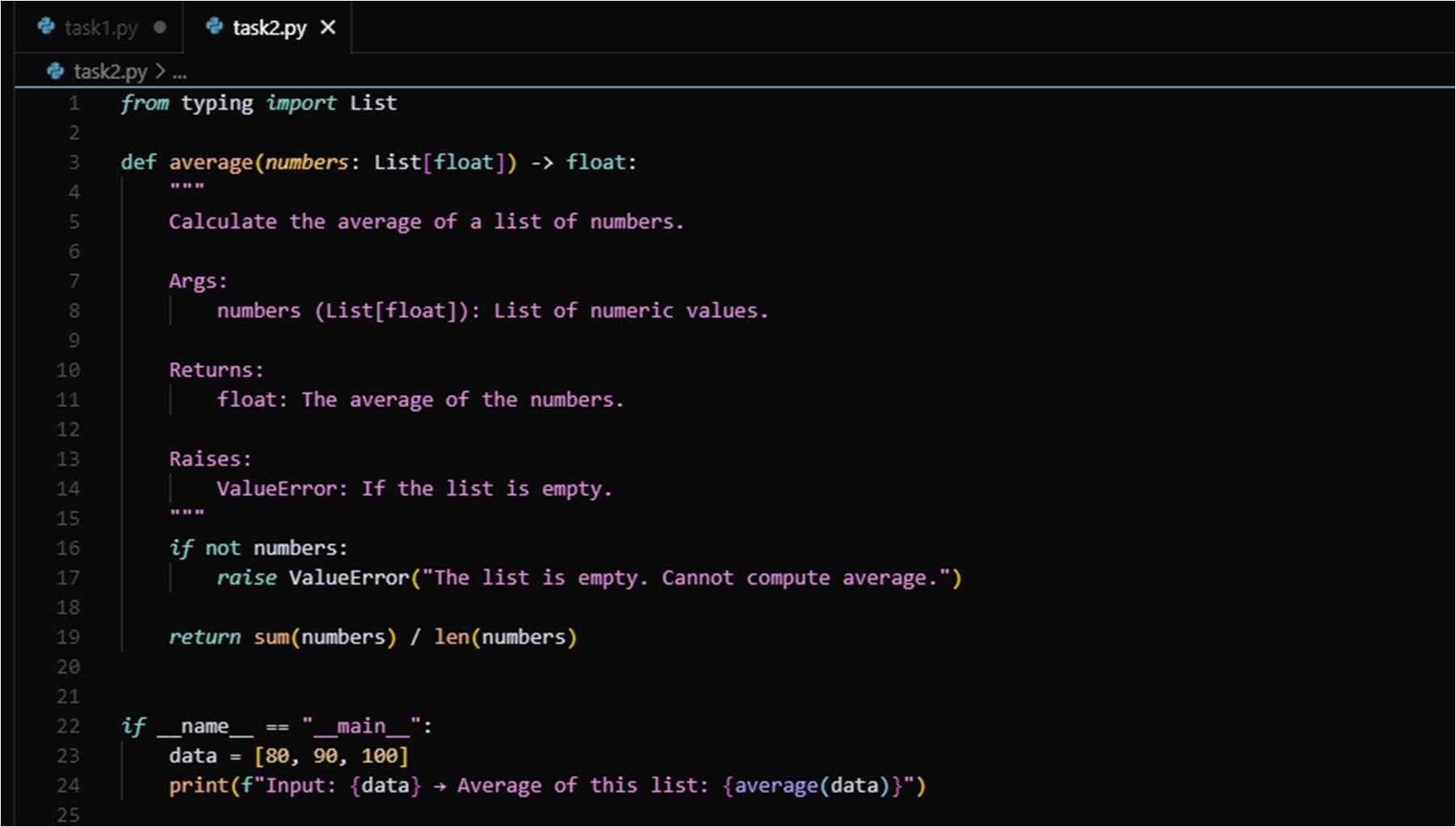
Input: [80, 90, 100] → Output: 90.0

## Acceptance Criteria:

* Self-explanatory function name.
* Handles empty input gracefully.

PROMPT: Refactor the given Python function to improve readability by using descriptive names and type hints, while ensuring it calculates the average of numbers correctly. Add error handling so it raises a ValueError if the input list is empty.

CORRECTED CODE:



OUTPUT:

