Coding & Development / Python Prompts

Write pytest unit tests for existing Python functions, including edge cases and mock examples.
Difficulty: Intermediate → Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Unit Testing, Test Automation, Code Quality
Updated: May 2026
Why This Prompt Exists
Most developers write code but skip tests — leaving bugs for production.

You get:

- code that works on happy path but fails on edge cases
- no tests for error conditions
- no mock examples for external dependencies
- tests that are hard to run or maintain
- low test coverage (untested code)

But unit tests are not optional.

They are the safety net for refactoring and deployment.

- Normal case: typical inputs, expected outputs
- Edge cases: empty lists, None values, boundary conditions
- Error cases: invalid inputs, exceptions raised
- Mock examples: external API calls, database queries
- Parametrized tests: test multiple inputs with one function

Without tests, you don’t know if your code works.

This framework forces AI to generate comprehensive unit tests.

The Prompt

```
Assume the role of a quality assurance engineer who writes thorough
pytest unit tests.

Your task is to generate unit tests for a Python function.

Generate:

1. IMPORTS
   - pytest
   - The function to test
   - unittest.mock (if mocking needed)

2. NORMAL CASE TESTS (2-3)
   - Typical inputs
   - Expected outputs

3. EDGE CASE TESTS (2-3)
   - Empty inputs, None values, boundary conditions
   - Single-item collections, zero values

4. ERROR CASE TESTS (2-3)
   - Invalid input types
   - Exceptions raised

5. MOCK TESTS (if external dependencies)
```

```
   - Mock API calls
   - Mock database queries
   - Mock file I/O

6. PARAMETRIZED TESTS (optional)
   - Test multiple inputs with @pytest.mark.parametrize

INPUTS:

Function Code (paste the function to test):
[PASTE FUNCTION CODE]

External Dependencies (if any):
[E.G., "Calls requests.get()", "Reads from file", "Queries database"]

Edge Cases to Test (if known):
[LIST OR "INFER"]

Test Framework:
[PYTEST / UNITTEST]

Coverage Goal:
[STATEMENT / BRANCH / PATH]

RULES:
- Test normal case first (happy path)
- Test edge cases (boundaries, empty, None)
- Test error cases (invalid inputs, exceptions)
- Use pytest fixtures for setup/teardown
```

```
- Use unittest.mock for external dependencies
- Use parametrize for testing multiple inputs
- Test names should describe what they test
- Assert one thing per test (when possible)
```

How To Use It

- Paste the exact function code — imports and all.
- List external dependencies so the prompt knows what to mock.
- Run pytest after generation and fix any failures (some may need adjustment).
- Add tests for any custom logic not covered.
- Aim for statement coverage (not perfection) as a starting point.

Example Input

Function Code:
def fetch_user_data(user_id, api_client):
response = api_client.get(f”/users/{user_id}”)
if response.status_code == 200:
return response.json()
elif response.status_code == 404:
return None
else:
raise Exception(f”API error: {response.status_code}”)

External Dependencies: Calls api_client.get() (needs mocking)

Edge Cases to Test: user_id not found (404), API returns 500 error

Test Framework: PYTEST

Why It Works

Most code has no tests.

This framework improves outcomes by forcing:

- normal case testing (baseline)
- edge case testing (robustness)
- error case testing (resilience)
- mock examples (isolation)
- parametrized tests (efficiency)

Great unit tests don’t just prove code works — they prove it fails gracefully when it should.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, Python frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The Python Data Analysis Script