Coding & Development / Python Prompts

Write Python functions from natural language descriptions, including docstrings, type hints, and error handling.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Code Generation, Python Development, Automation
Updated: May 2026
Why This Prompt Exists
Most developers waste time writing boilerplate functions from scratch when they could describe intent.

You get:

- functions missing docstrings (hard to maintain)
- no type hints (IDE can’t help with autocomplete)
- poor error handling (crashes on edge cases)
- inconsistent naming conventions
- functions that work but aren’t reusable

But a function is not just code.

It is a reusable unit with documentation, type safety, and error handling.

- Docstring: what it does, args, returns, raises
- Type hints: input and output types
- Error handling: try/except for edge cases
- Default values: sensible defaults where appropriate

Without documentation and error handling, functions are fragile.

This framework forces AI to generate production-ready functions.

The Prompt

```
Assume the role of a senior Python developer who writes clean,
documented, production-ready functions.

Your task is to generate a Python function from a natural language
description.

Generate:

1. FUNCTION SIGNATURE
   - Function name (descriptive, snake_case)
   - Parameters with type hints
   - Return type hint

2. DOCSTRING (Google or NumPy style)
   - Brief description
   - Args section with types and descriptions
   - Returns section
   - Raises section (if applicable)

3. FUNCTION BODY
   - Input validation
   - Main logic
   - Error handling (try/except)

4. EXAMPLE USAGE
```

```
   - How to call the function
   - Expected output

5. UNIT TEST (pytest)
   - Test normal case
   - Test edge case
   - Test error case

INPUTS:

Function Description:
[WHAT SHOULD THE FUNCTION DO?]

Input Parameters (names and types):
[LIST]

Return Value (type and description):
[DESCRIBE]

Edge Cases to Handle (if any):
[LIST]

Error Conditions to Handle:
[E.G., "File not found," "Invalid input," "Empty list"]

RULES:
- Use snake_case for function names
- Include type hints for all parameters and return
- Docstring must include Args, Returns, Raises
```

```
- Handle edge cases gracefully (not crash)
- Use descriptive variable names
- Follow PEP 8 style guide
```

How To Use It

- Be specific about input types (list of strings, dict with specific keys).
- List edge cases you know will happen (empty lists, None values).
- The generated unit test is a starting point — expand it with your own cases.
- Run the function through a linter (pylint, flake8) after generation.
- Add logging for functions that will run in production.

Example Input

Function Description: Calculate the average of a list of numbers. Return 0 if the list is empty.

Input Parameters: numbers (list of floats or ints)

Return Value: float (the average)

Edge Cases to Handle: Empty list, list with one item, list with negative numbers

Error Conditions to Handle: None (if input is not a list, raise TypeError)

Why It Works
Most generated functions lack documentation and error handling.

This framework improves outcomes by forcing:

- type hints (IDE support)
- docstrings (maintainability)
- error handling (robustness)

- unit tests (correctness)
- example usage (usability)

Great Python functions don’t just work — they’re documented, tested, and handle errors gracefully.

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 API Wrapper Builder