

Python Code Review — `calculator.py`

Skill: `.claude/skills/py-code-reviewer.md` **Reviewed:** `calculator.py` **Date:** 2026-02-20

Step 1 — Overview

Item	Detail
Purpose	Beginner-friendly interactive CLI calculator supporting addition and subtraction
Entry point	<code>if __name__ == "__main__" → main()</code> (line 50–51)
Dependencies	None — uses only Python built-in functions (<code>input</code> , <code>print</code> , <code>float</code>)
Module size	51 lines, 5 functions

The script presents a looping menu, accepts an operation choice, reads two numbers, computes the result, and prints it. Loops until the user enters `q`.

Step 2 — Structure & Readability

Functions and responsibilities:

Function	Responsibility
<code>add(a, b)</code>	Pure arithmetic — returns <code>a + b</code>
<code>subtract(a, b)</code>	Pure arithmetic — returns <code>a - b</code>
<code>get_number(prompt)</code>	Input loop — validates and returns a float
<code>show_menu()</code>	Displays the operation menu
<code>main()</code>	Orchestrates the full interactive loop

Findings:

#	Line	Issue	Recommendation
1	19–23	<code>show_menu()</code> uses 5 separate <code>print()</code> calls	Acceptable for readability; a single <code>print("".join(...))</code> would be marginally cleaner but is not required
2	37–38	Alignment spaces in prompts (<code>"Enter first number: "</code>) are intentional but undocumented	Add a brief inline comment if the alignment is intentional

No PEP 8 violations. All lines are well within 88 characters. Naming is clear and consistent throughout. No global variables. No unnecessary abstractions.

Rating: 5 / 5

Step 3 — Correctness & Logic

Findings:

#	Line	Observation
1	1–2	<code>add(a, b)</code> correctly returns <code>a + b</code> for all numeric inputs including floats and negatives
2	5–6	<code>subtract(a, b)</code> correctly returns <code>a - b</code>
3	9–15	<code>get_number</code> retries indefinitely on invalid input and returns the first valid float — correct behaviour
4	33–47	<code>main</code> handles all three branches (<code>q</code> , <code>+/-</code> , invalid) with no unreachable paths
5	13	Returns <code>float</code> — whole-number input <code>2</code> is returned as <code>2.0</code> . Cosmetically noticeable in output (<code>2.0 + 3.0 = 5.0</code>) but arithmetically correct
6	42, 45	f-string display shows floats as-is; <code>1.0 + 2.0 = 3.0</code> may surprise users who typed integers — not a bug, a UX note

No logic bugs, off-by-one errors, or unreachable code paths found.

Rating: 5 / 5

Step 4 — Pythonic Style

Findings:

#	Line	Pattern	Assessment
1	10	<code>while True + return</code> inside <code>try</code>	Correct Python idiom for input-retry loops <input checked="" type="checkbox"/>
2	11	<code>.strip()</code> applied to <code>input()</code> result	Good defensive habit <input checked="" type="checkbox"/>
3	14	Catches only <code>ValueError</code> , not bare <code>except</code>	Correct narrow exception handling <input checked="" type="checkbox"/>
4	36	<code>choice in ("+", "-")</code> membership test	Idiomatic <input checked="" type="checkbox"/>
5	40–45	<code>if/else</code> inside validated <code>elif</code> branch	Clean dispatch logic <input checked="" type="checkbox"/>
6	50	<code># pragma: no cover</code> on <code>__main__</code> guard	Correct coverage exclusion <input checked="" type="checkbox"/>

No anti-patterns found. Code is fully idiomatic for its scope and beginner-friendly.

Rating: 5 / 5

Step 5 — Type Hints & Documentation

Type hint coverage: None — no function has parameter or return annotations.

Docstring coverage: None — no function has a docstring.

Missing annotations:

```
# Current
def add(a, b):
    return a + b

# Recommended
def add(a: float, b: float) -> float:
    """Return the sum of a and b."""
    return a + b
```

```
# Current
def subtract(a, b):
    return a - b

# Recommended
def subtract(a: float, b: float) -> float:
    """Return the difference of a and b (a - b)."""
    return a - b
```

```
# Current
def get_number(prompt):
    ...

# Recommended
def get_number(prompt: str) -> float:
    """Prompt the user repeatedly until a valid float is entered, then return it."""
    ...
    ...
```

```
# Current
def show_menu():
    ...

# Recommended
def show_menu() -> None:
    """Print the calculator operation menu."""
    ...
    ...
```

```

# Current
def main():
    ...

# Recommended
def main() -> None:
    """Run the interactive calculator loop until the user quits."""
    ...

```

Rating: 2 / 5 — All functionality is correct, but zero type hints and zero docstrings means the code is not self-documenting for tooling (mypy, IDE autocomplete, help()).

Step 6 — Security Concerns

#	Severity	Line	Finding	Recommendation
1	<input checked="" type="checkbox"/> NONE	11	<code>input()</code> result passed only to <code>float()</code>	Safe — <code>float()</code> rejects non-numeric strings with <code>ValueError</code>
2	<input checked="" type="checkbox"/> NONE	31	Menu choice compared by string equality only	Safe — no code execution from user input
3	<input checked="" type="checkbox"/> NONE	All	No <code>eval</code> , <code>exec</code> , <code>subprocess</code> , <code>os.system</code>	Clean
4	<input checked="" type="checkbox"/> NONE	All	No hardcoded secrets, passwords, or API keys	Clean
5	<input checked="" type="checkbox"/> NONE	All	No file I/O, network calls, or external dependencies	Clean
6	<input checked="" type="checkbox"/> NONE	All	No use of <code>pickle</code> , <code>yaml.load()</code> , or unsafe deserializers	Clean

No security issues found.

Rating: 5 / 5

Step 7 — Performance Observations

#	Line	Observation
1	10	<code>while True</code> input-retry loop exits immediately on valid input — not a busy-wait
2	29	Main <code>while True</code> loop has no redundant computation inside it
3	19–23	Five <code>print()</code> calls in <code>show_menu()</code> — negligible cost for a CLI tool
4	All	No data structures, copies, or collections used — nothing to optimise

No performance issues for this scope.

Rating: 5 / 5

Step 8 — Testability

#	Function	Testability	Notes
1	<code>add</code>	Excellent	Pure function — no mocking needed
2	<code>subtract</code>	Excellent	Pure function — no mocking needed
3	<code>get_number</code>	Good	Requires <code>unittest.mock.patch("builtins.input")</code>
4	<code>show_menu</code>	Good	Requires <code>capsys</code> or <code>unittest.mock.patch("builtins.print")</code>
5	<code>main</code>	Excluded	<code># pragma: no cover</code> — correct decision for integration-level function

Suggested high-value unit test cases:

1. `test_add_negative_numbers` — verify `add(-3, -2) == -5`
2. `test_get_number_invalid_then_valid` — mock `input` to return "abc" first, then "5", assert return value is `5.0`
3. `test_get_number_float_string` — mock `input` to return "3.14", assert return value is `3.14`
4. `test_show_menu_output` — capture stdout and assert it contains "Simple Calculator", "+", "-", and "q"

Rating: 4 / 5 — Pure functions are trivially testable; `main()` is reasonably excluded; `show_menu()` stdout capture test would close the last coverage gap.

Step 9 — Summary Scorecard

Category	Rating (1-5)	Key Finding
Structure	5	Small, focused functions; no global state
Correctness	5	No logic bugs; edge cases handled correctly
Pythonic Style	5	Idiomatic loops, membership tests, exception handling
Type Hints & Docs	2	No type annotations or docstrings on any function
Security	5	No unsafe operations; safe input handling throughout
Performance	5	No inefficiencies for this scope
Testability	4	Pure functions easy to test; I/O correctly excluded

Overall: 4.4 / 5

Rating guide: 1 = needs major work · 3 = acceptable · 5 = excellent

Step 10 — Prioritised Action List

1. **[MEDIUM]** Add type hints to all 5 functions — lines 1, 5, 9, 18, 26
 2. **[MEDIUM]** Add one-line docstrings to `add`, `subtract`, `get_number`, `show_menu`, `main` — improves IDE support and `help()` output
 3. **[LOW]** Consider formatting float results with `f"{{result:g}}`" to display `3` instead of `3.0` for whole-number inputs — cosmetic UX improvement
 4. **[LOW]** Add a `show_menu` stdout capture test to close the last coverage gap
-

This is a disciplined, beginner-friendly script that correctly follows the CLAUDE.md architecture spec. The main growth area is adding type annotations and docstrings to bring it to production-grade readability — the core logic needs no changes.