Python Code Review — calculator.py

Version: v1.0.0
Date: 2026-02-20
Reviewer: Claude Code (automated structured review)
Files reviewed: calculator.py (51 lines) · tests/test_calculator.py (130 lines)

Step 1 — Overview

calculator.py is a beginner-friendly, single-file CLI calculator. It accepts user input interactively, performs addition or subtraction on two floating-point numbers, and loops until the user quits.

Entry point: `if __name__ == "__main__": main()` (line 50). The `# pragma: no cover` marker is correctly placed.

Dependencies: None — only Python builtins used.

Function	Responsibility
add(a, b)	Returns the sum of two numbers
subtract(a, b)	Returns the difference of two numbers
get_number(prompt)	Loops until user enters a valid float
show_menu()	Prints operation menu to stdout
main()	Main REPL loop; dispatches to add/subtract

Step 2 — Structure & Readability

The code is well-structured and readable. Functions are small and single-purpose, variable names are clear, and indentation is PEP 8 compliant. All lines stay within the 88-character limit.

2.1 — No docstrings on any function (lines 1, 5, 9, 18, 26)

Every public function is missing a docstring. Without them, help(calculator.add) returns nothing useful.

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

# After
def add(a, b):
    """Return the sum of a and b."""
    return a + b
```

2.2 — No type annotations on any function (lines 1, 5, 9, 18, 26)

Function signatures carry no type information. Readers and tools (mypy, IDEs) cannot infer parameter or return types.

```
# Before
def get_number(prompt):

# After
def get_number(prompt: str) -> float:
```

Step 3 — Correctness & Logic

The logic is correct for all standard inputs. Two minor edge cases exist that do not affect correctness per se but may surprise beginners.

3.1 — float() silently accepts 'inf', 'nan', '-inf' (line 13) [LOW]

Python's float() accepts IEEE 754 special values. A user who types 'inf' passes validation and produces confusing but valid output (inf + 1.0 = inf). Optional guard using math.isfinite() is recommended for a beginner tool.

3.2 — No KeyboardInterrupt handling (lines 10, 29) [LOW]

Pressing Ctrl+C raises KeyboardInterrupt uncaught, printing a traceback. Wrapping main() in try/except KeyboardInterrupt gives a friendlier exit.

```
try:
    while True:
        ...
except KeyboardInterrupt:
    print('\nGoodbye!')
```

Positive: No off-by-one errors, no unreachable code, correct ValueError guard, .strip() applied to all user input.

Step 4 — Pythonic Style

The code is clean and idiomatic. None of the common anti-patterns are present.

Anti-pattern	Present?	Note
range(len(...)) loop	No	—
Mutable default argument	No	—
String concatenation in loop	No	f-strings used correctly
Bare except clause	No	except ValueError is specific
Redundant type checks	No	—
Global variables	No	All state passed as arguments

Step 5 — Type Hints & Documentation

Function	Parameters typed?	Return typed?
add(a, b)	No	No
subtract(a, b)	No	No
get_number(prompt)	No	No
show_menu()	No	No
main()	No	No

Recommended complete annotations:

```
def add(a: float, b: float) -> float: ...
def subtract(a: float, b: float) -> float: ...
def get_number(prompt: str) -> float: ...
def show_menu() -> None: ...
def main() -> None: ...
```

No function has a docstring. All five public functions should have at minimum a one-line description of purpose, parameters, and return value.

Step 6 — Security Concerns

No security concerns for a local CLI tool of this scope. Bandit static analysis found zero issues.

#	Severity	Line	Finding
1	LOW	13	float() accepts 'inf'/'nan' — not a vulnerability, unguarded special values

No eval(), exec(), subprocess, os.system, pickle, yaml.load(), hardcoded secrets, or unsafe file-path construction present. Input is parsed via float() — safe.

Step 7 — Performance Observations

No performance concerns. Code is minimal and straightforward. show_menu() calls print() four times — consolidating into one multi-line print() is cosmetically cleaner but has negligible runtime impact.

Step 8 — Testability

Excellent. The test suite achieves 100% branch coverage on calculator.py. add and subtract are pure functions — ideal for unit testing. get_number isolates input() behind a parameter. main() terminates on 'q', making the REPL fully testable. capsys and monkeypatch fixtures are used correctly.

Suggested additional test cases (not gaps — coverage is 100%):

```
# 1. Whitespace-only input rejected
def test_get_number_whitespace_then_valid(monkeypatch, capsys):
    inputs = iter(['   ', '3'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    assert calculator.get_number('Enter: ') == 3.0

# 2. Negative numbers in main addition path
def test_main_addition_with_negatives(monkeypatch, capsys):
    inputs = iter(['+', '-3', '-4', 'q'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    calculator.main()
    assert '-3.0 + -4.0 = -7.0' in capsys.readouterr().out
```

Step 9 — Summary Scorecard

Category	Rating (1-5)	Key Finding
Structure	4 / 5	Well-organised; missing docstrings
Correctness	4 / 5	All paths correct; inf/nan edge unguarded
Pythonic Style	5 / 5	Clean, idiomatic — no anti-patterns
Type Hints & Docs	2 / 5	No type annotations or docstrings anywhere
Security	5 / 5	Bandit clean; no vulnerabilities
Performance	5 / 5	Trivial code; no inefficiencies
Testability	5 / 5	100% coverage, fixtures used correctly

OVERALL	**4.3 / 5**	**—**

Step 10 — Prioritised Action List

Priority	Severity	Finding	Location
1	MEDIUM	Add type hints to all five function signatures	Lines 1,5,9,18,26
2	MEDIUM	Add docstrings to all five public functions	Lines 1,5,9,18,26
3	LOW	Guard get_number against inf/nan special floats	Line 13
4	LOW	Handle KeyboardInterrupt in main() for graceful Ctrl+C exit	Line 29
5	INFO	Consolidate show_menu() into a single multi-line print()	Lines 19-23

This is clean, well-tested, beginner-friendly code that meets its stated goals. Adding type hints and docstrings is the single most impactful next step — it will make the module self-documenting and IDE-friendly with minimal effort. Well done!