Code Explanation: calculator.py

Version: v1.0.0 | Date: 2026-02-20 | File: calculator.py

1. Analogy — The Vending Machine

Think of this calculator like a **vending machine**:

- The **menu screen** (show_menu) shows what is available.
- You **press a button** (choice = input(...)) to pick an operation.
- The machine **asks for your coins** (get_number) — and keeps asking if you put in a banana instead of a coin.
- It **dispenses the result** (add or subtract) and displays it.
- Press the **exit button** (q) to walk away — or keep using it, the loop runs until you quit.

2. Program Flow Diagram

```
      +-------------------------------+
      |           main()              |
      |  'Welcome to Calculator'      |
      +----------+--------------------+
                 |
                 v
      +-------------------------------+
+---->|       show_menu()             |<-----------------+
|     |   Prints +, -, q options      |                  |
|     +----------+--------------------+                  |
|                v                                       |
|     +-------------------------------+                  |
|     |   choice = input(...)         |                  |
|     +----------+--------------------+                  |
|                |                                       |
|        +-------+-----------------------+               |
|        v                               v               |
|  choice == 'q'            choice == '+' or '-'         |
|        |                               |               |
|        v                               v               |
|  print('Goodbye!')          get_number(prompt)         |
|  break <-- EXIT           a = float,  b = float        |
|                                       |               |
|                           +-----------+---------+      |
|                           v                     v      |
|                       add(a,b)          subtract(a,b)  |
|                           +-----------+---------+      |
|                               print result             |
+-----------------------------------------------------------+
                      (loop again)
```

3. Step-by-Step Code Walkthrough

add(a, b) — Lines 1–2

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

The simplest possible function. Takes two numbers, returns their sum. Pure math, no side effects.

subtract(a, b) — Lines 5–6

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

Same shape as add. Subtracts b from a. Pure and simple.

get_number(prompt) — Lines 9–15

```
def get_number(prompt):
    while True:
        user_input = input(prompt).strip()
        try:
            return float(user_input)
        except ValueError:
            print('  Invalid number. Please try again.')
```

Line	What it does
while True:	Loop forever until a valid number is entered
input(prompt).strip()	Show prompt, read text, remove surrounding whitespace
return float(user_input)	Convert text to decimal; return and exit loop if successful
except ValueError:	Catch conversion failures (e.g., user typed 'abc')
print('Invalid...')	Notify user and loop back to ask again

show_menu() — Lines 18–23

```
def show_menu():
    print('\n--- Simple Calculator ---')
    print('  + : Addition')
    print('  - : Subtraction')
    print('  q : Quit')
    print('-------------------------')
```

Pure display code. Prints the menu each loop. The \n adds a blank line for visual breathing room between iterations.

main() — Lines 26–47

```
def main():
    print('Welcome to the Simple Calculator!')

    while True:
        show_menu()
        choice = input('Choose an operation: ').strip()

        if choice == 'q':
            print('Goodbye!')
            break
        elif choice in ('+', '-'):
            a = get_number('Enter first number:  ')
            b = get_number('Enter second number: ')

            if choice == '+':
                result = add(a, b)
                print(f'  {a} + {b} = {result}')
            else:
                result = subtract(a, b)
                print(f'  {a} - {b} = {result}')
        else:
            print('  Invalid option. Please choose +, -, or q.')
```

- **1.** Greet the user once.
- **2.** Enter a while True loop — runs until the user quits.
- **3.** Show menu, then read a choice.
- **4a.** 'q' → say goodbye, break, program ends.
- **4b.** '+' or '-' → ask for two numbers, compute, display.
- **4c.** Anything else → tell the user the input was invalid, loop again.

if __name__ == "__main__": — Lines 50–51

```
if __name__ == "__main__":  # pragma: no cover
    main()
```

Only runs main() when the file is executed directly — not when imported by the test suite. The # pragma: no cover tells the coverage tool to skip this line during automated testing.

4. Gotcha — Float Output Surprises

When you enter whole numbers like **1** and **2**, the result prints as **1.0 + 2.0 = 3.0** — not **3**.

Why? Because get_number converts all input to float. So '1' becomes 1.0, and add(1.0, 2.0) returns 3.0.

To display integers cleanly, you could write:

```
# Display as int if result is a whole number
display = int(result) if result == int(result) else result
```

But this is intentionally left simple — float handles both integers and decimals without two separate code paths. Clarity over cosmetics.

Summary

Function	Role	Key Pattern
add	Pure math	Simple return
subtract	Pure math	Simple return
get_number	Input validation	while True + try/except

show_menu	Display	Side-effect only (print)
main	Orchestrator	while True + branching

The code follows a clean **separation of concerns**: math functions do not do I/O, input functions do not do math, and main() ties everything together.