Code Explanation: calculator.py

Version: v1.1.0 | Date: 2026-02-20 | File: calculator.py

1. Real-Life Analogies

Analogy 1 — Starbucks Order Kiosk

Think of the calculator as a **Starbucks self-service kiosk**:

Kiosk Step	Calculator Equivalent
Kiosk displays the menu	show_menu() prints options
You tap Hot / Cold Drinks	You type + or -
Kiosk asks Size? Milk?	get_number() prompts for numbers
Kiosk rejects invalid input	get_number() loops on non-numeric input
Receipt prints your total	print(f'{a} + {b} = {result}')
You press New Order	while True loop restarts show_menu()
You press Done	You type q to quit

Analogy 2 — Airport Self-Check-in Terminal

The **main loop** mirrors an airport kiosk:

- Screen refreshes with options after every interaction (show_menu() at top of loop).
- Wrong flight number re-prompts you (the else branch prints 'Invalid option').
- Choosing 'Check In' starts a sub-flow for seat/bags — like entering two numbers after +/-.
- Cancel / Done exits cleanly — the q branch.

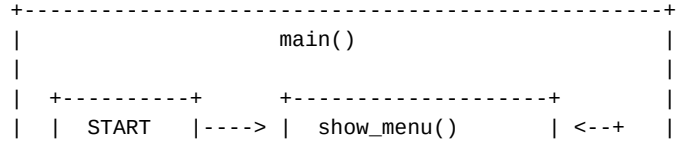
Analogy 3 — Grocery Store Self-Checkout

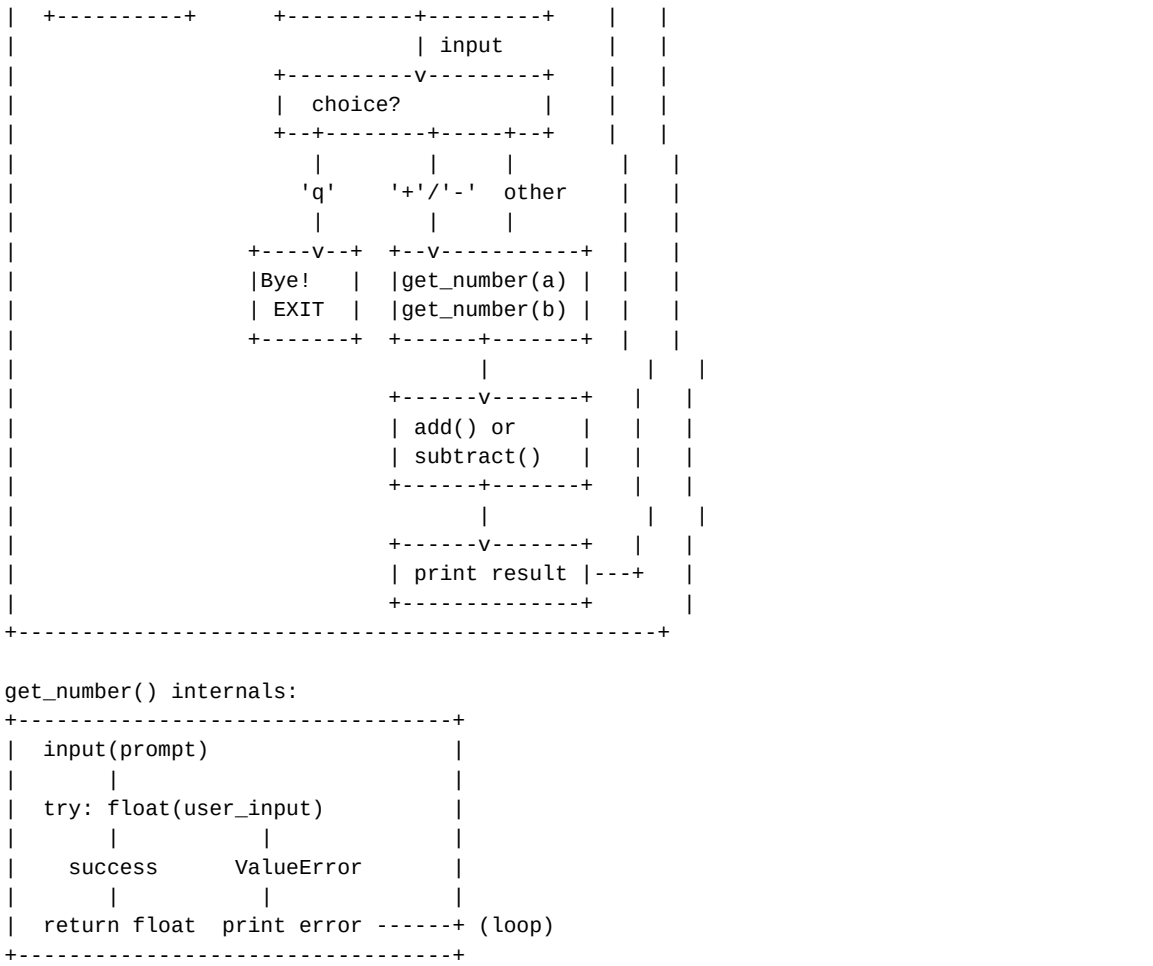
get_number() is the **barcode scanner**:

- You scan an item (type a number).
- If the barcode won't read (invalid input), the machine beeps 'Try again' — the except ValueError block.
- Only a valid scan (parseable float) lets you move on.

---

2. High-Level Flow — ASCII Diagram





---

3. Step-by-Step Code Walkthrough

Lines 1-2 add(a, b)

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

Pure function. Takes two values, returns their sum. No side effects, no state. Testable in isolation.

Lines 5-6 subtract(a, b)

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

Subtracts b from a. Named functions (vs inline math) make them independently unit-testable without running the UI.

Lines 9-15 get_number(prompt)

```
def get_number(prompt):
    while True:
```

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

- **while True** — infinite retry loop; only exits via return.
- **.strip()** — removes accidental leading/trailing whitespace.
- **float()** — converts the string. 'abc' raises ValueError; '3.14' and '5' both succeed.
- **except ValueError** — catches bad input, prints friendly message, loops again.

Lines 18-23 show_menu()

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

Display-only. The \n adds a blank line above for readability. Isolating this means the menu text is editable in one place.

Lines 26-47 main()

```
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.')
```

- Prints welcome once, then enters an infinite loop.
- Each iteration shows the menu and reads a choice.
- Three branches: q (exit with break), +/- (compute and print), else (invalid — re-prompt).
- choice in ('+', '-') checks tuple membership — readable and efficient.
- f-strings format the result inline.

Lines 50-51 Entry-Point Guard

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

__**name**__ is "__main__" only when run directly. When imported by tests it equals "calculator" — so main() does NOT auto-run. # **pragma: no cover** excludes this line from coverage measurement.

---

4. Gotcha — The float() Trap

Common misconception: "My calculator only needs integers, so I should use int() instead of float()."

- int("3.14") raises ValueError — int() does not accept decimal strings.
- float("3") works fine, returning 3.0.
- Using float() handles both 5 and 5.5 with no extra logic.

Second gotcha: The while True / return pattern in get_number() looks odd to beginners. There is no loop counter or flag — the loop just runs until a valid float is returned.

Third gotcha: choice == "Q" would NOT match. The code uses lowercase "q". A capital Q hits the "Invalid option" branch. Adding .lower() on the input would fix this if needed.

---

5. Summary Table

Function	Role	Side Effects?	Testable Alone?
`add(a, b)`	Math	No	Yes
`subtract(a, b)`	Math	No	Yes
`get_number(prompt)`	Input guard / retry loop	Yes (prints)	Yes (mock input)
`show_menu()`	Display	Yes (prints)	Yes (capture stdout)
`main()`	Orchestrator / event loop	Yes (full I/O)	Yes (mock I/O)
