Horrible Code Activity

Program: Command-Line Calculator (Python)

Date: February 20, 2026

Files included:

good_code_calculator.py (recommended practices) and **bad_code_calculator.py** (intentional violations).

Overview

This activity demonstrates how coding style and structure impact readability, maintainability, and correctness. Both versions implement the same feature: the user types an expression like **10 / 2** and the program prints the result. The **good** version follows best practices; the **bad** version intentionally breaks them.

How to Run

```
On macOS / Linux:

python3 good_code_calculator.py
python3 bad_code_calculator.py
```

Both programs support operators: **+**, **-**, *****, **/**. Type **q** to quit.

Selected Principles

Principle	What it means (short)
KISS	Keep it simple: clear control flow, minimal surprise.
DRY	Don't Repeat Yourself: avoid copy/paste; reuse logic via functions or data structures.
Single Responsibility / Separation of Concerns	Each function/module should do one job; keep input/output separate from business logic.

What the Good Version Does

KISS: small, predictable functions (add/subtract/multiply/divide). Main loop is short and readable.

DRY: uses a single **OPERATIONS** dictionary to map symbols to functions, avoiding repeated if/elif logic.

Single Responsibility & Separation of Concerns:

• **parse_request()** validates and converts user input to numbers.

• **compute()** performs calculation only.

• **main()** handles user interaction (printing, input loop).

Documentation: docstrings and clear naming help a reader understand the code quickly.

Errors: meaningful error messages (e.g., divide by zero) without crashing the program.

What the Bad Version Does (Intentionally)

Not KISS: vague names (doStuff), random global state, noisy prints, confusing control flow.

Not DRY: copy/paste blocks for each operator; same parsing and printing repeated multiple times.

Not Single Responsibility / Not Separation of Concerns: one big function mixes input, parsing, math, output, and debugging side-effects.

Poor error handling: bare **except** hides real problems; unclear messages like “nope”.

Incorrect behavior: division by zero prints a nonsense number instead of a proper error.

YAGNI (extra): imports math even though it isn't used (dead code).

Quick Comparison

Area	Good code	Bad code (intentional issues)
Structure	Multiple small functions + clear main()	Single large function with mixed responsibilities
Repetition	Operation dispatch table (DRY)	Copy/paste per operator (not DRY)
Readability	Consistent naming + docstrings	Vague names + noisy output + globals
Error Handling	Specific messages + safe failures	Bare except + unclear messages

Notes for Grading

Both files implement the same basic calculator feature and include at least three calculation functions. The differences are in how the code is organized, named, documented, and how repetition and responsibilities are handled.