Good Coding Practices

- **KISS (Keep It Simple, Stupid)**
  - The code avoids unnecessary complexity by using a clean structure with functions for each operation.
  - Uses a dictionary to map operations to functions instead of long if-else chains.
- **DRY (Don't Repeat Yourself)**
  - The get_number function is used to handle user input validation, eliminating repeated input checks.
  - Each mathematical operation is in its own function, avoiding redundant logic.
- **Single Responsibility Principle**
  - Each function is responsible for a single task: calculation, input handling, or program execution.
  - The main function manages user interaction, while separate functions handle calculations and input validation.
- **Separation of Concerns**
  - The code is modular, separating input handling, calculations, and main execution logic.
  - This makes it easier to maintain and modify without affecting other parts of the program.
- **Clean Code**
  - Proper naming for functions and variables improves readability.
  - Functions are well-structured and short, making the code easy to understand.
  - Error handling prevents crashes (i.e. handling division by zero).
- **Document Your Code**
  - Each function has a clear docstring explaining its purpose.
  - Proper comments guide users without excessive or redundant explanations.

Bad Coding Practices

- **KISS Violation**
  - Uses long, nested if-else statements instead of a simpler, more efficient approach.
  - Overly repetitive logic makes the code harder to read and modify.
- **DRY Violation**
  - Input validation is repeated multiple times for each operation instead of using a function.
  - Each operation duplicates the same input-checking logic instead of centralizing it.
- **Single Responsibility Violation**
  - The code mixes multiple concerns within the same blocks (e.g., handling input validation within each operation check).
  - No clear separation between different tasks, making modifications more error-prone.
- **Separation of Concerns Violation**
  - No modular functions; everything is written in a long, monolithic structure.
  - Difficult to maintain, as changes require modifying multiple places in the code.
- **Lack of Clean Code**
  - Unnecessary variable reassignments (i.e., reprocessing input unnecessarily).
  - Inconsistent formatting and redundant operations lowers readability.
  - No error handling for division by zero and some other specific cases, leading to potential crashes.
- **Lack of Documentation**
  - No comments or explanations for functions, making it hard to understand.
  - Variables and function names are not descriptive, reducing code clarity.