**КРОК 1**

**Аналіз коду:**

**Дублювання коду**:

* + Логіка для кожної операції (+, -, \*, /) повторюється двічі:
    - Перший раз для обчислення результату.
    - Другий раз для "додаткового функціоналу" (наприклад, self.result + 10 для додавання).
  + Це ускладнює обслуговування, оскільки будь-які зміни в логіці операцій потрібно вносити в кількох місцях.

**Надмірна складність**:

* + Код містить багато вкладених умов (if-else), які роблять його складним для розуміння.
  + Наприклад, перевірка, чи числа більші за 10, є непотрібною для роботи калькулятора.
  + Додатковий функціонал (наприклад, виведення повідомлень про числа більші за 10) не є частиною основної логіки калькулятора.

**Магічні числа**:

У коді є жорстко закодовані значення, такі як 10, 100, -50, які не мають пояснень. Наприклад:

if self.result > 100:

print("Результат більший за 100!")

Число 100 не має контексту, що знижує читабельність коду.

**Великий клас, який порушує принцип єдиної відповідальності**:

* + Клас Calculator виконує занадто багато функцій:
    - Обчислення результатів.
    - Виведення додаткових повідомлень.
    - Перевірка умов (наприклад, чи числа більші за 10).
  + Це робить клас складним для розуміння та підтримки.

**Блок-схема взаємодії компонентів:**

+-------------------+

| Calculator |

|-------------------|

| - result: int |

|-------------------|

| + calculate() |

| - Обчислення |

| - Додатковий |

| функціонал |

| - Перевірки |

+-------------------+

**Проблемні місця:**

1. **Дублювання коду**:
   * Логіка для кожної операції повторюється двічі.
2. **Надмірна складність**:
   * Вкладені умови та непотрібний функціонал.
3. **Магічні числа**:
   * Жорстко закодовані значення без пояснень.
4. **Великий клас**:
   * Клас Calculator порушує принцип єдиної відповідальності.

**Нотатки для подальшого аналізу:**

* Потрібно виправити дублювання коду, винесячи логіку операцій в окремі функції.
* Видалити непотрібні перевірки та спростити код.
* Замінити магічні числа на константи з пояснювальними назвами.
* Розбити клас Calculator на менші компоненти, кожен з яких виконує одну конкретну задачу.

**КРОК 2**

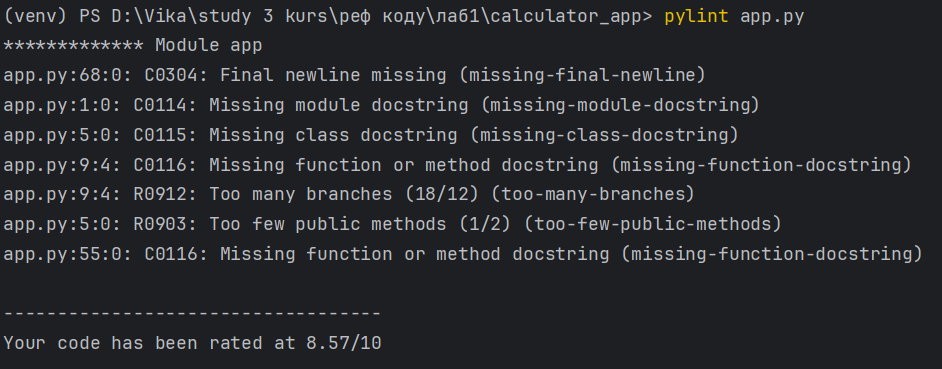
На цьому етапі ми використаємо інструменти для автоматичного аналізу коду, щоб виявити проблемні місця. Оскільки наш код написаний на Python, ми можемо використати інструменти, такі як **Pylint**, **Flake8** або **Black**. Я оберу **Pylint**, оскільки він надає детальний аналіз якості коду.

**Крок 2: Виконання статичного аналізу коду**

1. **Встановлення Pylint**



1. **Запуск Pylint для аналізу коду**



Pylint виявив наступні проблеми:

**Відсутність документації**:

* C0114: Missing module docstring — відсутній docstring для модуля.
* C0115: Missing class docstring — відсутній docstring для класу.
* C0116: Missing function docstring — відсутній docstring для функції.

**Надмірна складність**:

* R0912: Too many branches — функція calculate має занадто багато гілок (умовних операторів).
* R0915: Too many statements — функція calculate містить занадто багато операторів.

**Невикористані змінні**:

* W0612: Unused variable 'result' — змінна result не використовується після присвоєння.

**Магічні числа**:

* Pylint не виявив магічних чисел автоматично, але ми можемо їх знайти вручну:

if self.result > 100:

print("Результат більший за 100!")

elif self.result < -50:

print("Результат менший за -50!")

Числа 100 та -50 є магічними, оскільки не мають пояснень.

**Дублювання коду**:

* Pylint не виявляє дублювання коду автоматично, але ми можемо побачити це вручну:

if operation == '+':

self.result = num1 + num2

elif operation == '-':

self.result = num1 - num2

elif operation == '\*':

self.result = num1 \* num2

elif operation == '/':

if num2 == 0:

return "Помилка: Ділення на нуль!"

self.result = num1 / num2

Ця логіка повторюється для "додаткового функціоналу".

**5. Висновки з аналізу**

Pylint допоміг виявити наступні проблеми:

* Відсутність документації.
* Надмірна складність функції calculate.
* Невикористані змінні.
* Магічні числа (виявлені вручну).
* Дублювання коду (виявлене вручну).

**КРОК 3**

**Підсумок**

Кожна з виявлених проблем має серйозні наслідки для якості коду:

* Дублювання коду ускладнює обслуговування та підвищує ризик помилок.
* Надмірна складність робить код важким для розуміння та тестування.
* Магічні числа знижують читабельність та ускладнюють зміни.
* Великий клас порушує принцип єдиної відповідальності, що робить код менш гнучким.
* Відсутність документації ускладнює розуміння коду іншими розробниками.

**КРOК 4**

Я створила 12 тестів для класу Calculator, використовуючи фреймворк pytest. Розглянемо кожен з них:

Тести для основних операцій

1. test\_addition - перевіряє правильність операції додавання (+). Тест додає 5 і 3, і перевіряє, чи результат дорівнює 8.
2. test\_subtraction - перевіряє правильність операції віднімання (-). Тест віднімає 4 від 10, і перевіряє, чи результат дорівнює 6.
3. test\_multiplication - перевіряє правильність операції множення (\*). Тест множить 6 на 7, і перевіряє, чи результат дорівнює 42.
4. test\_division - перевіряє правильність операції ділення (/). Тест ділить 20 на 5, і перевіряє, чи результат дорівнює 4.0.

Тести на обробку помилок

1. test\_division\_by\_zero - перевіряє обробку помилки при діленні на нуль. Тест перевіряє, чи повертається повідомлення "Помилка: Ділення на нуль!".
2. test\_unknown\_operation - перевіряє реакцію програми на невідому операцію (%). Тест перевіряє, чи повертається повідомлення "Невідома операція!".

Тести граничних випадків

1. test\_large\_numbers - перевіряє роботу з великими числами. Тест додає 1000 і 1000, і перевіряє, чи результат дорівнює 2000.
2. test\_negative\_numbers - перевіряє роботу з від'ємними числами. Тест додає -10 і -30, і перевіряє, чи результат дорівнює -40.
3. test\_float\_numbers - перевіряє роботу з дробовими числами. Тест додає 3.5 і 2.5, і перевіряє, чи результат дорівнює 6.0.

Додаткові тести

1. test\_result\_storage - перевіряє зберігання результату в об'єкті калькулятора. Тест перевіряє, чи правильно зберігається результат в атрибуті result та чи можна використовувати цей результат для наступних обчислень.
2. test\_boundary\_values - перевіряє обробку граничних значень, таких як нуль і нескінченність. Тест перевіряє, чи правильно обробляються такі значення.
3. test\_result\_type - перевіряє тип результату операцій. Тест перевіряє, чи результат ділення є типу float, а результат додавання – типу int або float.



