Лабораторная работа: Основы модульного тестирования в Python

Инструкция

Вы будете работать в парах (группы по 2 человека).

Студент A — пишет unit-тесты (файл test_calculator.py).

Студент B — пишет реализацию функций (файл calculator.py).

В Google Colab вы будете имитировать разделение ролей, создавая два блока кода: один — для тестов, другой — для реализации.

Задание: Калькулятор с расширенными функциями

Реализуйте модуль calculator, содержащий следующие функции:

1. add(a, b) — сложение двух чисел.
2. subtract(a, b) — вычитание: a - b.
3. multiply(a, b) — умножение.
4. divide(a, b) — деление. Если b == 0, выбрасывает ValueError.
5. power(base, exponent) — возведение в степень. Поддерживает целые и вещественные числа.
6. is_even(n) — возвращает True, если n — чётное целое число, иначе False. Если n не целое — выбрасывает TypeError.

```
# test_calculator.py (Студент A)

import unittest

# Импортируем функции из calculator (пока они не реализованы — это нормально!)
# В Colab мы просто определим их позже, но тесты пишутся ДО реализации

class TestCalculator(unittest.TestCase):

    def test_add(self):
        # your code here
        pass

    def test_subtract(self):
        # your code here
        pass

    def test_multiply(self):
        # your code here
        pass

    def test_divide(self):
        self.assertEqual(divide(10, 2), 5)
        self.assertEqual(divide(7, 2), 3.5)
        with self.assertRaises(ValueError):
            divide(5, 0)

    def test_power(self):
        self.assertEqual(power(2, 3), 8)
```

```
        self.assertEqual(power(5, 0), 1)
        self.assertAlmostEqual(power(4, 0.5), 2.0, places=7)
        self.assertEqual(power(-2, 3), -8)

    def test_is_even(self):
        self.assertTrue(is_even(4))
        self.assertFalse(is_even(5))
        self.assertTrue(is_even(-2))
        self.assertFalse(is_even(0))  # 0 — чётное! Исправьте, если нужно
        with self.assertRaises(TypeError):
            is_even(3.5)
        with self.assertRaises(TypeError):
            is_even("4")

if __name__ == '__main__':
    unittest.main(argv=[''], verbosity=2, exit=False)
```

```
test_add (__main__.TestCalculator.test_add) ... ok
test_divide (__main__.TestCalculator.test_divide) ... ok
test_is_even (__main__.TestCalculator.test_is_even) ... ok
test_multiply (__main__.TestCalculator.test_multiply) ... ok
test_power (__main__.TestCalculator.test_power) ... ok
test_subtract (__main__.TestCalculator.test_subtract) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.014s

OK
```

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

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

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

def power(base, exponent):
    return base ** exponent

def is_even(n):
    if not isinstance(n, int):
        raise TypeError("Input must be an integer")
    return n % 2 == 0
```