Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Лабораторна роботи №6

з дисципліни «Спеціалізовані мови програмування»

на тему

«Розробка та Unit тестування Python додатку»

Виконав:

Костик В. Ю.

Перевірив:

Щербак С. С.

Львів 2024

**Мета:** Cтворення юніт-тестів для додатка-калькулятора на основі класів

**План роботи**

Завдання 1: Тестування Додавання

Напишіть юніт-тест, щоб перевірити, що операція додавання в вашому додатку-калькуляторі працює правильно. Надайте тестові випадки як для позитивних, так і для негативних чисел.

Завдання 2: Тестування Віднімання

Створіть юніт-тести для переконання, що операція віднімання працює правильно. Тестуйте різні сценарії, включаючи випадки з від'ємними результатами.

Завдання 3: Тестування Множення

Напишіть юніт-тести, щоб перевірити правильність операції множення в вашому калькуляторі. Включіть випадки з нулем, позитивними та від'ємними числами.

Завдання 4: Тестування Ділення

Розробіть юніт-тести для підтвердження точності операції ділення. Тести повинні охоплювати ситуації, пов'язані з діленням на нуль та різними числовими значеннями.

Завдання 5: Тестування Обробки Помилок

Створіть юніт-тести, щоб перевірити, як ваш додаток-калькулятор обробляє помилки. Включіть тести для ділення на нуль та інших потенційних сценаріїв помилок. Переконайтеся, що додаток відображає відповідні повідомлення про помилки.

**Реалізація:**

**Папка AppSettings, файл AppSettings.py:**  
decimal\_places = 2

**Папка classes, файл calculator.py:**

from GlobalVariables.GlobalVariables import memory, history

from functions.functions import get\_operator, calculate

from tools.tools import save\_to\_memory, get\_from\_memory, show\_history

from AppSettings.AppSettings import decimal\_places

class Calculator:

def \_\_init\_\_(self):

self.memory = memory

self.history = history

self.decimal\_places = decimal\_places

def get\_numbers(self):

choice = input("Використати значення з пам'яті? (так/ні): ").lower()

if choice not in ['так', 'ні']:

print("Помилка: введіть так або ні.")

return self.get\_numbers()

if choice == 'так':

num1 = get\_from\_memory()

if num1 is None:

return self.get\_numbers()

print(f"Використано значення з пам'яті: {num1}")

else:

try:

num1 = float(input("Введіть перше число: "))

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return self.get\_numbers()

try:

num2 = float(input("Введіть друге число: "))

return num1, num2

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return self.get\_numbers()

def run(self):

while True:

num1, num2 = self.get\_numbers()

operator = get\_operator()

result = calculate(num1, num2, operator)

if result is not None:

result = round(result, self.decimal\_places)

print(f"Результат: {result}")

self.history.append(f"{num1} {operator} {num2} = {result}")

self.ask\_save\_to\_memory(result)

if not self.ask\_continue():

break

if self.ask\_view\_history():

show\_history()

if self.ask\_customize():

self.customize()

def ask\_save\_to\_memory(self, result):

save\_choice = input("Зберегти результат у пам'ять? (так/ні): ").lower()

if save\_choice == 'так':

save\_to\_memory(result)

def ask\_continue(self):

return input("Чи хочете виконати ще одне обчислення? (так/ні): ").lower() == 'так'

def ask\_view\_history(self):

return input("Бажаєте переглянути історію? (так/ні): ").lower() == 'так'

def ask\_customize(self):

return input("Бажаєте налаштувати калькулятор? (так/ні): ").lower() == 'так'

def customize(self):

try:

new\_decimal\_places = int(input("Введіть кількість десяткових розрядів для результатів (за замовчуванням 2): "))

self.decimal\_places = new\_decimal\_places

print(f"Кількість десяткових розрядів змінена на: {self.decimal\_places}")

self.history.append(f"Користувач змінив кількість десяткових знаків на: {self.decimal\_places}")

except ValueError:

print("Помилка: потрібно ввести ціле число.")

self.history.append("Помилка при налаштуванні десяткових розрядів.")

**Папка functions, файл functions.py:**import math

def validate\_input(prompt, validation\_fn, error\_message):

while True:

user\_input = input(prompt)

if validation\_fn(user\_input):

return user\_input

print(error\_message)

def get\_operator():

return validate\_input("Введіть оператор (+, -, \*, /, %, ^, √): ",

lambda op: op in ['+', '-', '\*', '/', '%', '^', '√'],

"Помилка: некоректний оператор.")

def calculate(num1, num2, operator):

try:

operations = {

'+': lambda: num1 + num2,

'-': lambda: num1 - num2,

'\*': lambda: num1 \* num2,

'/': lambda: num1 / num2 if num2 != 0 else None,

'%': lambda: num1 % num2,

'^': lambda: num1 \*\* num2,

'√': lambda: math.sqrt(num1) if num1 >= 0 else None,

}

return operations.get(operator, lambda: None)()

except (ZeroDivisionError, ValueError) as e:

print(f"Помилка: {e}")

return None

**Папка GlobalVariables, файл GlobalVariables.py:**

memory = None

history = []

**Папка interface, файл interface.py:**

def get\_user\_input(prompt):

return input(prompt)

def display\_output(message):

print(message)

**Папка logs, файл logs.py:**

def log\_error(error\_message):

with open("error\_log.txt", "a") as log\_file:

log\_file.write(f"Помилка: {error\_message}\n")

def log\_calculation(expression, result):

with open("calculation\_log.txt", "a") as log\_file:

log\_file.write(f"Обчислення: {expression} = {result}\n")

**Папка tools, файл tools.py:**

from GlobalVariables.GlobalVariables import memory, history

def save\_to\_memory(result):

global memory

memory = result

print(f"Збережено у пам'яті: {memory}")

def get\_from\_memory():

if memory is None:

print("Помилка: пам'ять порожня.")

return None

return memory

def show\_history():

if history:

for i, entry in enumerate(history, 1):

print(f"{i}: {entry}")

else:

print("Історія порожня.")

**Файл main.py:**

import sys

import os

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

parent\_dir = os.path.abspath(os.path.join(current\_dir, '..'))

sys.path.append(parent\_dir)

from classes.calculator import Calculator

def main():

calculator = Calculator()

calculator.run()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Файл test\_calculator.py**import unittest

from classes.calculator import Calculator

from functions.functions import calculate

import sys

import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

class TestCalculatorOperations(unittest.TestCase):

def setUp(self):

# Ініціалізуємо калькулятор перед кожним тестом

self.calculator = Calculator()

# Завдання 1: Тестування Додавання

def test\_addition(self):

result = calculate(5, 3, '+')

self.assertEqual(result, 8, "Додавання не працює з позитивними числами")

result = calculate(-5, -3, '+')

self.assertEqual(result, -8, "Додавання не працює з від'ємними числами")

# Завдання 2: Тестування Віднімання

def test\_subtraction(self):

result = calculate(10, 5, '-')

self.assertEqual(result, 5, "Віднімання не працює з позитивними числами")

result = calculate(5, 10, '-')

self.assertEqual(result, -5, "Віднімання не працює для отримання від'ємного результату")

result = calculate(-5, -3, '-')

self.assertEqual(result, -2, "Віднімання не працює з від'ємними числами")

# Завдання 3: Тестування Множення

def test\_multiplication(self):

result = calculate(0, 5, '\*')

self.assertEqual(result, 0, "Множення не працює з нулем")

result = calculate(3, 3, '\*')

self.assertEqual(result, 9, "Множення не працює з позитивними числами")

result = calculate(-3, 3, '\*')

self.assertEqual(result, -9, "Множення не працює з від'ємними числами")

# Завдання 4: Тестування Ділення

def test\_division(self):

result = calculate(10, 2, '/')

self.assertEqual(result, 5, "Ділення не працює з позитивними числами")

result = calculate(10, 0, '/')

self.assertIsNone(result, "Ділення на нуль не обробляється правильно")

result = calculate(-10, 2, '/')

self.assertEqual(result, -5, "Ділення не працює з від'ємними числами")

# Завдання 5: Тестування Обробки Помилок

def test\_error\_handling(self):

# Тест на неіснуючий оператор

result = calculate(10, 2, '?')

self.assertIsNone(result, "Некоректний оператор не обробляється правильно")

# Тест ділення на нуль

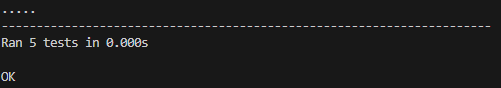
result = calculate(10, 0, '/')

self.assertIsNone(result, "Помилка ділення на нуль не обробляється правильно")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**Результат виконання (test\_calculator.py):**

****

**Висновок:** на цій лабораторній роботі я додав набір юніт-тестів, які перевіряють правильність основних арифметичних операцій у додатку-калькуляторі. Ці тести допоможуть виявити та виправити будь-які проблеми або помилки, які можуть виникнути під час розробки чи обслуговування вашого додатку, забезпечуючи його надійність і точність