МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Лабораторна робота №6

з дисципліни

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

на тему

РОЗРОБКА ТА UNIT ТЕСТУВАННЯ PYTHON ДОДАТКУ

Виконав:

ст. гр. РІ-31

ЛАЗАР В.С.

Прийняв:

ЩЕРБАК С.С.

Львів-2024

**Мета роботи:**

Створення юніт-тестів для додатка-калькулятора на основі класів.

**Хід роботи:**

**Завдання 1: Тестування Додавання**

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

**Завдання 2: Тестування Віднімання**

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

**Завдання 3: Тестування Множення**

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

**Завдання 4: Тестування Ділення**

Розробити 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**

Створити юніт-тести, щоб перевірити, як додаток-калькулятор обробляє помилки. Включити тести для ділення на нуль та інших потенційних сценаріїв помилок. Переконатися, що додаток відображає відповідні повідомлення про помилки.

**Код програми:**

"""The user interface of the lab work"""

import unittest

import global\_variables

from Data.Shared.functions.calculator import calculate

from Data.Shared.classes.history import History

from Data.Shared.classes.validators import Validators

from Data.Shared.classes.unit\_test import UnitTest

from Data.Shared.functions.logger import logger

class Console:

"""The console class of this lab work"""

instance = None

def \_\_new\_\_(cls):

if cls.instance is None:

cls.instance = super(Console, cls).\_\_new\_\_(cls)

return cls.instance

def \_\_init\_\_(self, digits = 3):

self.digits = digits

self.main()

def main(self):

"""The main menu of this lab work"""

while True:

case = input("\n1 - Calculate a number \n"

"2 - View history \n"

"3 - Additional settings \n"

"4 - Unit test \n"

"Your choice: ")

match case:

case "1":

logger.info("[Lab 6] Started performing calculation")

try:

self.calculator()

except ValueError as e:

print(e)

case "2":

logger.info("[Lab 6] Read history")

History.read()

case "3":

logger.info("[Lab 6] Opened setting menu")

self.settings()

case "4":

logger.info("[Lab 6] Started unit tests")

self.run\_unit\_tests()

case \_:

return

@staticmethod

def run\_unit\_tests():

"""Runs unit tests"""

print("Running unit tests...\n")

suite = unittest.defaultTestLoader.loadTestsFromTestCase(UnitTest)

runner = unittest.TextTestRunner()

runner.run(suite)

@staticmethod

def calculator():

"""Does all the verifications before calculating a number"""

num1 = Validators.validate\_num("\nEnter first number (or MR / MC): ")

operator = Validators.validate\_operator()

if operator in global\_variables.MEMORY\_OPERATIONS:

Validators.validate\_memory(operator, num1)

return False

num2 = Validators.validate\_num("Enter second number (or MR / MC): ")

try:

result = calculate(num1, num2, operator)

except ZeroDivisionError:

print("Error: cannot divide by zero")

return False

print("Result : " + str(result))

try\_again = input("\nCalculation has finished successfully! \n"

"Current options: \n"

"Try again? (Y / N) \n"

"Store a value into memory? (MS / M+ / M-) \n"

"Your choice: ").lower()

if try\_again in global\_variables.MEMORY\_OPERATIONS:

Validators.validate\_memory(try\_again, result)

return True

if try\_again == "y":

return False

return True

def settings(self):

"""Allows to change digits after a decimal point in a number or to clear history"""

settings\_prompt = input("\n1 - Change the amount of digits"

" after a decimal point in a number \n"

"2 - Clear history\n"

"Your choice: ")

match settings\_prompt:

case "1":

while True:

digits\_prompt = input("\nEnter the amount of digits (Current value: "

+ str(self.digits) + "): ")

try:

self.digits = Validators.validate\_digits(digits\_prompt)

global\_variables.DIGITS = self.digits

print("Settings changed successfully\n")

break

except ValueError as e:

print(e)

case "2":

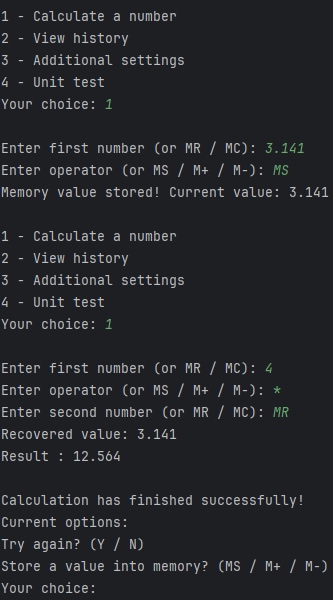
History.clear()

print("History cleared successfully")

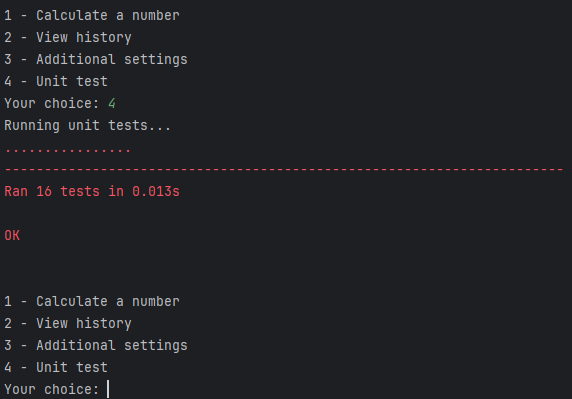
case \_:

print("Invalid input")

На рис. 1-2 зображено результат виконання програми:



*Рис. 1. Базова робота програми-калькулятора*

*  
Рис. 2. Приклад роботи юніт-тестів*

Посилання на Github: [PaperGlit/Python\_Lab\_6](https://github.com/PaperGlit/Python_Lab_6)

**Висновок:**

Виконавши ці завдання, у мене є набір юніт-тестів, які перевіряють правильність основних арифметичних операцій у моєму додатку-калькуляторі. Ці тести допомагають виявити та виправити будь-які проблеми або помилки, які можуть виникнути під час розробки чи обслуговування мого додатку, забезпечуючи його надійність і точність