МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Інститут комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та мереж

Лабораторна робота №1
з дисципліни: «Спеціалізовані мови програмування»
на тему: «Введення в Python»

Виконала:
студентка групи ІТ-32
Моляща Ю.А.
Прийняв:
Щербак С.С.

Львів - 2023

Лабораторна робота №1

«Введення в Python»

Мета роботи: створення консольної програми-калькулятора за допомогою основних синтаксичних конструкцій Python, з іншим завданням на заміну тестуванню та валідації.

Завдання на лабораторну роботу.

Завдання 1. Введення користувача Створіть Python-програму, яка приймає введення користувача для двох чисел і оператора (наприклад, +, -, *, /).

Завдання 2. Перевірка оператора Перевірте чи введений оператор є дійсним (тобто одним із +, -, *, /). Якщо ні, відобразіть повідомлення про помилку і попросіть користувача ввести дійсний оператор.

Завдання 3. Обчислення Виконайте обчислення на основі введення користувача (наприклад, додавання, віднімання, множення, ділення) і відобразіть результат.

Завдання 4. Повторення обчислень Запитайте користувача, чи він хоче виконати ще одне обчислення. Якщо так, дозвольте йому ввести нові числа і оператор. Якщо ні, вийдіть з програми.

Завдання 5. Обробка помилок реалізуйте обробку помилок для обробки ділення на нуль або інших потенційних помилок. Відобразіть відповідне повідомлення про помилку, якщо виникає помилка.

Завдання 6. Десяткові числа Змініть калькулятор так, щоб він обробляв десяткові числа (плаваючу кому) для більш точних обчислень.

Завдання 7. Додаткові операції Додайте підтримку додаткових операцій, таких як піднесення до степеня (^), квадратний корінь (√) і залишок від ділення (%).

Завдання 8. Функція пам'яті Реалізуйте функцію пам'яті, яка дозволяє користувачам зберігати і відновлювати результати. Додайте можливості для зберігання та отримання значень з пам'яті.

Завдання 9. Історія обчислень Створіть журнал, який зберігає історію попередніх обчислень, включаючи вираз і результат. Дозвольте користувачам переглядати історію своїх обчислень.

Завдання 10. Налаштування користувача Надайте користувачам можливість налаштувати поведінку калькулятора, таку як зміну кількості десяткових розрядів, які відображаються, або налаштування функцій пам'яті.

Хід роботи.

Код програми:

main.py

```
# Import functions from console_calculator.py
from history_handling import load_history, save_history
from console_calculator import calculate_option, settings_option
# Import global variables
from history_handling import HISTORY_FILE
from console_calculator import number_of_calculations
def main():
    # Use global variable to change value outside of function
    global number_of_calculations

    while True:
        print("\nOptions:")
        print("1. Perform calculation")
        print("2. Settings")
        print("3. Quit")
        choice = input("Enter your choice (1/2/3): ")
        if choice == '1':
            history = load_history(HISTORY_FILE)
            result_dict = calculate_option()
            if number_of_calculations is not None:
                if len(history) >= number_of_calculations:
                    history.pop(0)
            history.append(result_dict)
            save_history(HISTORY_FILE, history)
        elif choice == '2':
            settings_option()
        elif choice == '3':
            print("Exiting the calculator. Goodbye!")
            break
        else:
            print("Invalid choice. Please select a valid option (1/2/3).")
if __name__ == "__main__":
    main()
```

console_calculator.py

```
from calculations import calculate
from input_handling import get_input
from history_handling import display_history, clear_history
from history_handling import HISTORY_FILE
# Default values
decimal_places = 2
number_of_calculations = None
# Main functions
```

```
def calculate_option():
    """
    Calculate the result of a mathematical operation and create a dictionary.
    Returns:
        result_dict (dict): Dictionary containing the operation and the
formatted result.
    """
    # Get input
    first_number, second_number, operator = get_input()
    # Calculate result
    result = calculate(first_number, second_number, operator)
    # Format result to decimal places
    formatted_result = format_result(result, decimal_places)
    # Create dictionary
    result_dict = format_dict(first_number, second_number, operator,
formatted_result)
    # Print result
    print("Result:", formatted_result)

    return result_dict
def settings_option():
    """Display settings options and perform the selected action."""
    while True:
        print("\nOptions:")
        print("1. Display calculation history")
        print("2. Clear history")
        print("3. Set memory functions")
        print("4. Set decimal places")
        print("5. Back")

        choice = input("Enter your choice (1/2/3/4/5): ")

        if choice == '1':
            display_history(HISTORY_FILE)
        elif choice == '2':
            clear_history(HISTORY_FILE)
        elif choice == '3':
            set_memory_functions()
        elif choice == '4':
            set_decimal_places()
        elif choice == '5':
            return
        else:
            print("Invalid choice. Please select a valid option (1/2/3/4/5).")
# Format functions
def format_result(result, decimal_places):
    """
    Format result to decimal places.
```

```
    Args:
        result (float): Result of calculation
        decimal_places (int): Number of decimal places to format result to

    Returns:
        formatted_result (str): Formatted result
    """
    return f"{result:.{decimal_places}f}"
def format_dict(first_number, second_number, operator, formatted_result):
    """
    Format result to dictionary.

    Args:
        first_number (float): First number
        second_number (float): Second number
        operator (str): Operator
        formatted_result (str): Formatted result

    Returns:
        result_dict (dict): Dictionary containing the operation and the
formatted result.
    """
    # Create empty dictionary
    result_dict = {}

    # Format operation for special cases
    if operator == '√':
        result_dict['operation'] = f'{operator} {first_number}'
    else:
        result_dict['operation'] = f'{first_number} {operator} {second_number}'

    # Format result
    result_dict['result'] = formatted_result

    return result_dict
# Settings functions
def set_decimal_places():
    """Set decimal places option."""
    # Use global variable to change value outside of function
    global decimal_places

    while True:
        try:
            decimal_places = int(input("Enter decimal places: "))
            if decimal_places >= 0:
                print(f"Decimal places set to {decimal_places}.")
                break
            else:
                print("Please enter a non-negative integer.")
```

```
        except ValueError:
            print("Invalid input. Please enter a valid integer.")

def set_memory_functions():
    """Display memory functions options and perform the selected action."""
    print("1. Auto-delete history")
    print("2. Back")
    choice = input("Enter your choice: ")

    if choice == '1':
        set_auto_delete_history()
    elif choice == '2':
        return
    else:
        print("Invalid choice. Please try again.")

def set_auto_delete_history():
    """Set auto-delete history option."""
    # Use global variable to change value outside of function
    global number_of_calculations

    while True:
        print("1. On")
        print("2. Off")
        print("3. Back")
        choice = input("Enter your choice (1/2): ")
        if choice == '1':
            while True:
                try:
                    number_of_calculations = int(input("Enter the number of
calculations to store: "))
                    print("Auto-delete history is on.")
                    break
                except ValueError:
                    print("Invalid input. Please enter a valid number.")
            break
        elif choice == '2':
            number_of_calculations = None
            print("Auto-delete history is off.")
            break
        elif choice == '3':
            return
        else:
            print("Invalid choice. Please select a valid option (1/2).")
```

input_handling.py

```
VALID_OPERATORS = ['+', '-', '*', '/', '^', '√', '%']
# Input functions
```

```
def get_input():
    """
    Get input from user
    Returns:
        first_number (float): First number
        second_number (float): Second number
        operator (str): Operator
    """
    while True:
        operator = input("Enter an operator (+, -, *, /, ^, √, %): ")
        if check_operator(operator):
            break
    # Special case for square root and power operators
    if operator == '^':
        first_number = get_numeric_input("Enter number: ")
        second_number = get_numeric_input("Enter power: ")
    elif operator == '√':
        first_number = get_numeric_input("Enter number: ")
        second_number = None
    # For all other operators
    else:
        first_number = get_numeric_input("Enter first number: ")
        second_number = get_numeric_input("Enter second number: ")
    return first_number, second_number, operator
def check_operator(operator):
    """
    Check if operator is valid.

    Args:
        operator (str): Operator
    """
    if operator not in VALID_OPERATORS:
        print("Invalid operator. Please try again.")
        return False
    else:
        return True

def get_numeric_input(prompt):
    """
    Get numeric input from user.

    Args:
        prompt (str): Prompt to display to user.
    """
    while True:
        try:
            value = float(input(prompt))
            return value
        except ValueError:
```

```
            print("Invalid input. Please enter a numeric value.")
```

calculations.py

```
from input_handling import get_input
def addition(first_number, second_number):
    return first_number + second_number
def subtraction(first_number, second_number):
    return first_number - second_number
def multiplication(first_number, second_number):
    return first_number * second_number
def division(first_number, second_number):
    if second_number == 0:
        print("Division by zero is not allowed. Enter values again.")
        first_number, second_number = get_input()
        return division(first_number, second_number)
    else:
        return first_number / second_number
def power(first_number, second_number):
    return first_number ** second_number
def square_root(first_number):
    if first_number < 0:
        print("Square root of negative number is not allowed. Enter value
again.")
        first_number = get_input()
        return square_root(first_number)
    return first_number ** 0.5
def remainder(first_number, second_number):
    if second_number == 0:
        print("Division by zero is not allowed. Enter values again.")
        first_number, second_number, operator = get_input()
        return remainder(first_number, second_number)
    return first_number % second_number
def calculate(first_number, second_number, operator):
    if operator == '+':
        return addition(first_number, second_number)
    elif operator == '-':
        return subtraction(first_number, second_number)
    elif operator == '*':
        return multiplication(first_number, second_number)
    elif operator == '/':
        return division(first_number, second_number)
    elif operator == '^':
        return power(first_number, second_number)
    elif operator == '√':
        return square_root(first_number)
    elif operator == '%':
        return remainder(first_number, second_number)
    else:
```

```
        return "Invalid operator. Please try again."
```

history_handling.py

```
HISTORY_FILE = 'source/lab1/history.txt'
# History functions
def save_history(HISTORY_FILE, history):
    """
    Saves history to file.

    Args:
        HISTORY_FILE (str): Path to history file.
        history (list): List of dictionaries containing the operation and the
result.
    """
    with open(HISTORY_FILE, "w") as file:
        for entry in history:
            file.write(f"{entry['operation']} = {entry['result']}\n")
def load_history(HISTORY_FILE):
    """
    Loads history from file.

    Args:
        HISTORY_FILE (str): Path to history file.

    Returns:
        history (list): List of dictionaries containing the operation and the
result.

    Raises:
        FileNotFoundError: If file does not exist.
    """
    history = []
    try:
        with open(HISTORY_FILE, "r") as file:
            for line in file:
                operation, result = line.strip().split(" = ")
                result_dict = {'operation': operation, 'result': float(result)}
                history.append(result_dict)
    except FileNotFoundError:
        pass
    return history
def display_history(HISTORY_FILE):
    """
    Display history from file.

    Args:
        HISTORY_FILE (str): Path to history file.
    """
```

```
    history = load_history(HISTORY_FILE)
    if len(history) == 0:
        print("History is empty")
    else:
        with open(HISTORY_FILE, "r") as file:
            for line in file:
                print(line.strip())

def clear_history(HISTORY_FILE):
    """
    Clear history from file.

    Args:
        HISTORY_FILE (str): Path to history file.
    """
    with open(HISTORY_FILE, "w") as file:
        file.write("")
    print("History cleared")
```

Приклад виконання зображено на рисунках 1-3.

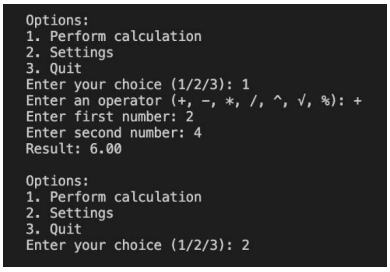

Рис. 1: Приклад виконання операції додавання

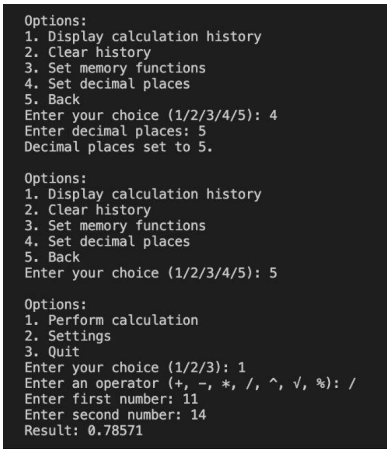

Рис. 2: Приклад використання функції зміни кількості десяткових розрядів

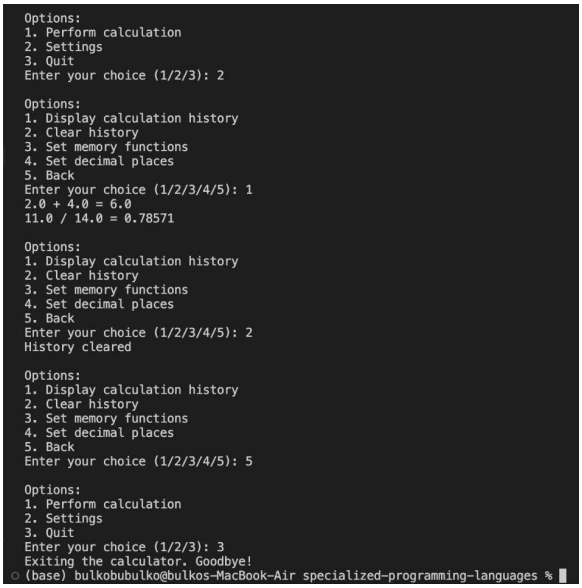

Рис. 3: Приклад використання функцій для роботи з пам’яттю

Висновок: під час виконання лабораторної роботи було створено консольну програму-калькулятор за допомогою основних синтаксичних конструкцій Python, з іншим завданням на заміну тестуванню та валідації.