Programowanie Strukturalne

-

Laboratorium 04 - Python lab01

Adam Szajgin s319821 gr. 105

Politechnika Warszawska

12 grudnia 2023

1.1 Temat programu

Poniższy program to proste narzędzie demonstracyjne umożliwiające użytkownikowi wykonywanie różnorodnych działań matematycznych oraz manipulację listami. Stworzone z myślą o prezentacji różnych funkcji, program oferuje możliwość wyboru operacji dodawania, mnożenia, porównywania i modyfikacji list poprzez prosty interfejs wiersza poleceń.

1.2 Opis programu

Program został zaprojektowany w celu eksponowania różnorodnych funkcji oraz operacji dostępnych w języku programowania python. Użytkownik ma wybór spośród czterech głównych operacji:

- Dodawanie: Służy do łączenia liczb lub ciągów znaków, oraz przy użyciu funkcji lambda dodanie wprowadzonych przez użytkownika liczb,

- Mnożenie: Pozwala na pomnożenie liczb oraz powielanie ciągów znaków,

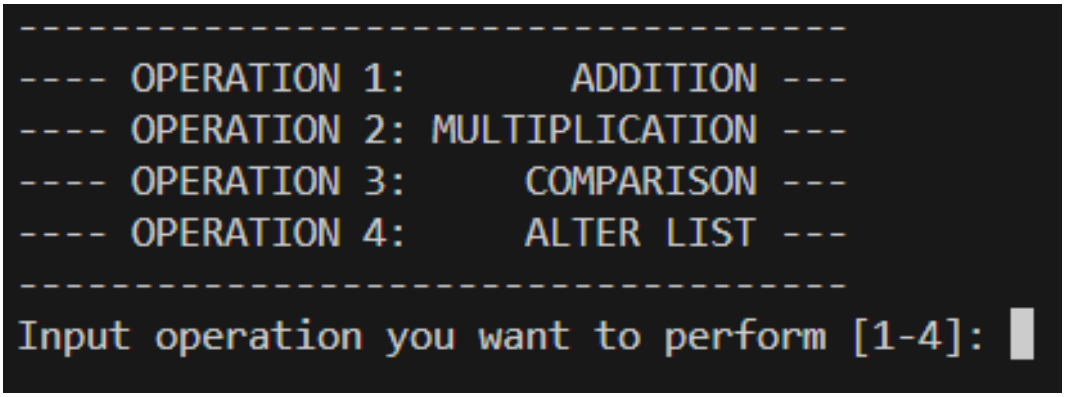
- Porównywanie: Umożliwia porównanie dwóch wartości, zarówno liczb, jak i ciągów znaków,

- Modyfikacja listy: Daje możliwość modyfikowania dwóch predefiniowanych list: listy owoców i listy samochodów. Użytkownik może wybierać elementy do zmiany oraz określać ich nowe wartości.

Ten program został utworzony w celu demonstracyjnym, dlatego też wykorzystałem przykładowe dane, aby pokazać działanie różnych funkcji. Nie był optymalizowany pod kątem uniwersalności czy efektywności.

Również do każdej z funkcji dodałem opcję wywołania help(*funkcja*) tak by użytkownik z pozycji terminala mógł sprawdzić co robi i jak działa dana funkcja. W postaci "fragmentu z dokumentacji".

Po uruchomieniu programu, użytkownik napotyka menu wyboru operacji w terminalu:



Rys. 1. Menu programu

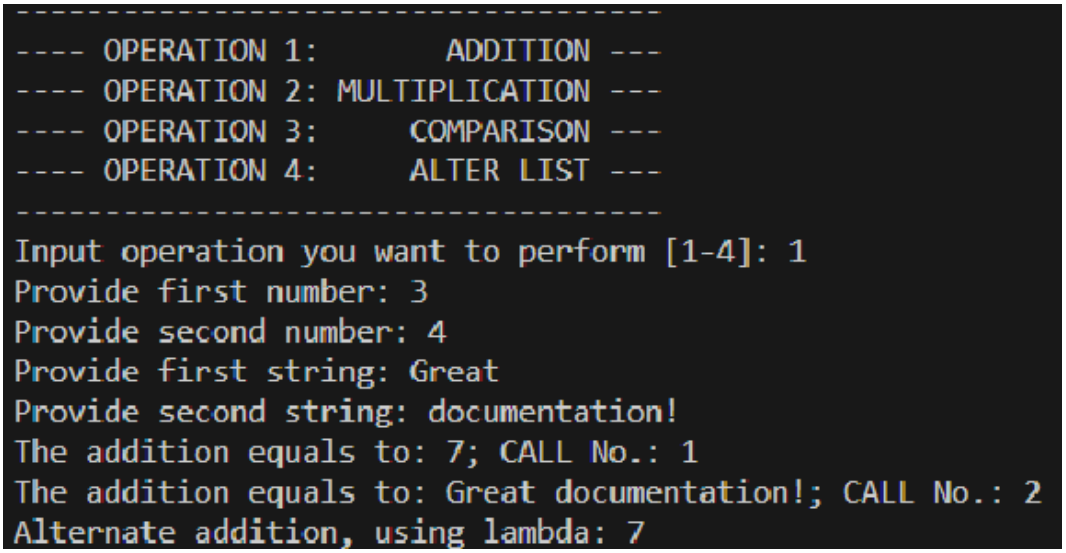
A poniżej część kodu odpowiedzialna za wydruk menu, pobieranie numeru wybranej operacji oraz inicjalizację zmiennej globalnej, później wykorzystanej przy opcji pierwszej - ADDITION, do okreslania numeru wywołania:

```
# Making the terminal UI a bit prettier and intuitive
# The user chooses the operation and the numbers/strings
# they want to perform the operation on

print("----------")
print("---- OPERATION 1: ADDITION ---")
print("---- OPERATION 2: MULTIPLICATION ---")
print("---- OPERATION 3: COMPARISON ---")
print("---- OPERATION 4: ALTER LIST ---")
print("----------")
op_choice = int(input("Input operation you want to perform [1-4]: "))

# initializing the later called global variable
j = 0
```

Rys. 2. Kod odpowiedzialny za print menu oraz inicjację globalnej zmiennej

1. Test pierwszej opcji programu - dodawania.:

   Pierwsza liczba: 3. Druga liczba: 4. Pierwszy string: "Great ". Drugi string: "Documentation!



Rys. 3. Symulacja wyboru opcji: ADDITION

Część kodu odpowiedzialna za dodawanie i kolejna za wywołanie:

```
# function for addition, it accepts any kind of input, but is being tested on intigers and strings
def addition(a, b):
    """This function adds up two input arguments, either a string or an integer!"""
    global j
    j += 1
    addition_result = a + b
    print("The addition equals to: {}; CALL No.: {}".format(addition_result, j))
```

Rys. 4. funkcja ADDITION

```python
# Typical for structural programming, switch case structure, wanted to add this to
match op_choice:
    # First case, adding up the input, testing for intiger and then a string
    case 1:
        X = int(input("Provide first number: "))
        Y = int(input("Provide second number: "))
        Z = str(input("Provide first string: "))
        G = str(input("Provide second string: "))

        addition(X, Y)
        addition(Z, G)
        print(f"Alternate addition, using lambda: {(lambda a, b: a + b)(X, Y)} ")
```

Rys. 5. wywołanie funkcji ADDITION

Tutaj warto zwrócić uwagę na funkcję lambda i jej wykorzystanie. Można ją zakodować w miejscu wywołania, jest to szczególnie przydatne, gdy wiemy, że tej funkcji użyjemy tylko raz i w danym miejscu!

2. Test drugiej opcji programu - mnożenia.

   Pierwsza liczba: 4. Druga liczba: 7. Pierwszy string: ”Hi! ”. Trzecia liczba: 4

```
-------------------------------------
---- OPERATION 1:       ADDITION ---
---- OPERATION 2: MULTIPLICATION ---
---- OPERATION 3:     COMPARISON ---
---- OPERATION 4:     ALTER LIST ---
-------------------------------------
Input operation you want to perform [1-4]: 2
Provide first number: 4
Provide second number: 7
Provide a string: Hi!
Provide sthe number by which you want to multiply that string: 4
The multiplication equals to: 28
The multiplication equals to: Hi! Hi! Hi! Hi!
```

Rys. 6. Symulacja wyboru opcji: MULTIPLICATION

Część kodu odpowiedzialna za mnożenie i kolejna za wywołanie:

```python
# function for multiplication, it accepts any kind of input, but is being tested on intigers and strings
def multiplication(a, b):
    """This function multiplies up two input arguments, either a string or an integer!
    In case of integers, simply multiplies them, in case of string, it multiplies it by
    the number later provided by the user"""
    multiplication_result = a * b
    print("The multiplication equals to: {}".format(multiplication_result))
```

Rys. 7. funkcja MULTIPLICATION

```python
# Second case, multiplying the input, testing for intiger and then a string
case 2:
    X = int(input("Provide first number: "))
    Y = int(input("Provide second number: "))
    Z = str(input("Provide a string: "))
    G = int(
        input("Provide sthe number by which you want to multiply that string: ")
    )

    multiplication(X, Y)
    multiplication(Z, G)
```

Rys. 8. wywołanie funkcji MULTIPLICATION

3. Test trzeciej opcji programu - porównania.
   Pierwsza liczba: 6. Druga liczba: -10. Pierwszy string: "hi" . Drugi string: "hello"

```
-------------------------------------
---- OPERATION 1:       ADDITION ---
---- OPERATION 2: MULTIPLICATION ---
---- OPERATION 3:     COMPARISON ---
---- OPERATION 4:     ALTER LIST ---
-------------------------------------
Input operation you want to perform [1-4]: 3
Provide first number: 6
Provide second number: -10
Provide first string: hi
Provide second string: hello
The 6 is bigger than -10
The length of hi is smaller than hello
```

Rys. 9. Symulacja wyboru opcji: COMPARISON

W tym przypadku jednak, sytuacja jest trochę inna. Funkcja mimo, że akceptuje dowolny argument (nie tylko int i string) to rozróżnia instancję gdy dostaje stringa, jako, że ich porównywanie rozgrywa się nieco inaczej niż zwykłych liczb czy innych argumentów.
Wciąż, ćwiczenie polega na implementacji funkcji o różnych typach argumentów, a to owa funkcja spełnia.
Część kodu odpowiedzialna za porównanie i kolejna za wywołanie:

```python
# function for comparison, it accepts any kind of input, but is being tested on intigers and strings
def comparing(a, b):
    """ "This function compares which is greater (or bigger) of two input arguments,
    either a string or an integer!"""

    if isinstance(a, str) and isinstance(b, str):
        if len(a) < len(b):
            print("The length of {} is smaller than {}".format(a, b))
        else:
            print("The length of {} is bigger or equal to {}".format(a, b))
    else:
        if a < b:
            print("The {} is smaller than {}".format(a, b))
        elif a == b:
            print("The {} is equal to {}".format(a, b))
        else:
            print("The {} is bigger than {}".format(a, b))
```

Rys. 10. funkcja COMPARISON

```python
# Third case, comparing the input, testing for intiger and then a string
case 3:
    X = int(input("Provide first number: "))
    Y = int(input("Provide second number: "))
    Z = str(input("Provide first string: "))
    G = str(input("Provide second string: "))

    comparing(X, Y)
    comparing(Z, G)
```

Rys. 11. wywołanie funkcji COMPARISON

4. Test czwartej opcji programu - zmiany w liście.
   Ta funkcja działa inaczej od reszty. Nie jest ona jedną z funkcji kalkulatora, a bardziej funkcją ”extra” stworzoną w celu pokazania ”funkcji której argumentem jest lista”.

   W tym przypadku w celach demonstarcyjnych ”zhardcodowałem” dwie listy, ”cars” oraz ”fruits”. W ramach działania programu, użytkownik może którą listę chce zmienić i wybrać zmianę jaką chcą w ramach danej listy wykonać.

   Funkcja najpierw drukuje listę w jej pierwotnym stanie i w kolejnej linijce zmienioną listę przez użytkownika.

```
-----------------------------------
---- OPERATION 1:       ADDITION ---
---- OPERATION 2: MULTIPLICATION ---
---- OPERATION 3:     COMPARISON ---
---- OPERATION 4:     ALTER LIST ---
-----------------------------------
Input operation you want to perform [1-4]: 4
Which list you want to change? (fruits/cars): cars
Give me the thing you want to change to: -TEST-
Give me the index of your change: 1
['Audi', 'Peugeot', 'Fiat']
['Audi', '-TEST-', 'Fiat']
```

Rys. 12. Symulacja wyboru opcji: ALTER LIST

Część kodu odpowiedzialna za zmianę w liście i kolejna za wywołanie:

```python
# function for altering a list, the user firstly chooses which of the two
# lists they want to alter, then chooses what they want to change to,
# and where in the list they want to make this change!
# It prints the default chosen list and then the altered version
def alter_list(some_list: list, index, new_value):
    print(some_list)
    some_list[index] = new_value
    print(some_list)
```

Rys. 13. funkcja ALTER LIST

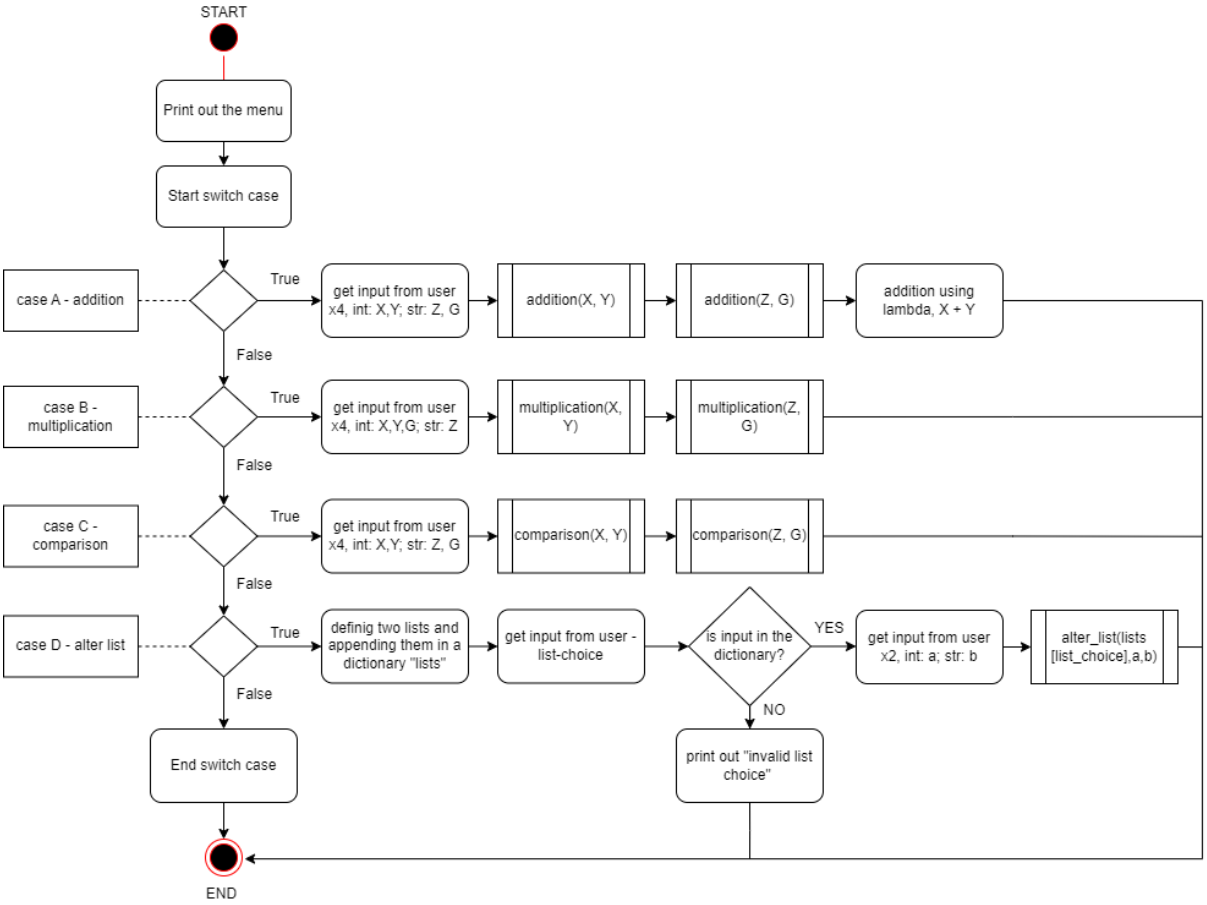
```python
# Fourth case, altering a specified list
case 4:
    # Predifinig two lists, im aware of this being a hardcode, but its just an example so yeah...
    fruits = ["apple", "banana", "cherry"]
    cars = ["Audi", "Peugeot", "Fiat"]

    # Mapping the input string to the actual list variable
    lists = {"fruits": fruits, "cars": cars}

    # Taking data from the user, then calling the function to do the specified operation
    list_choice = str(input("Which list you want to change? (fruits/cars): "))
    if list_choice in lists:
        b = str(input("Give me the thing you want to change to: "))
        a = int(input("Give me the index of your change: "))

        # Calling the alter_list function with, lists[list_choice] - from dictionary
        # "lists", choosing the one fitting list_choice,
        # altering with input a and b provided by the user.
        alter_list(lists[list_choice], a, b)
    else:
        print("Invalid list choice.")
```

Rys. 14. wywołanie funkcji ALTER LIST

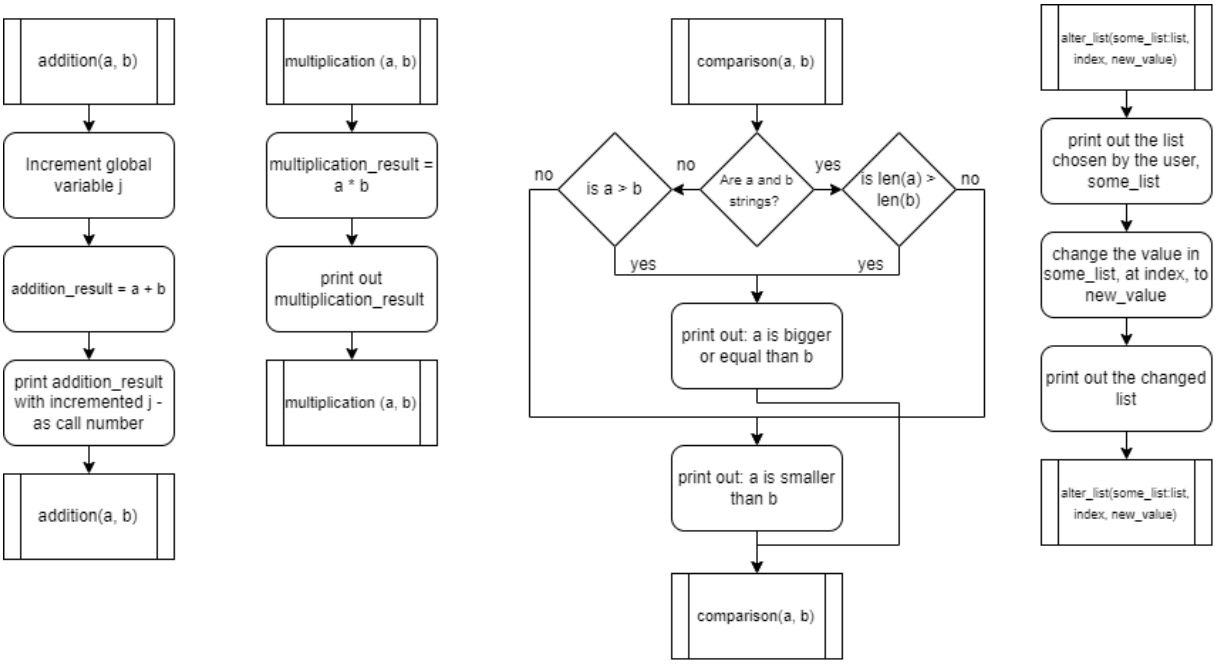
W tym przypadku, predefiniowałem dwie listy: 'fruits' (owoce) i 'cars' (samochody), a następnie mapowałem ich nazw do zmiennych w słowniku. Umożliwia to użytkownikowi wybór listy poprzez wpisanie nazwy ('fruits' lub 'cars'). Po dokonaniu wyboru użytkownik podaje nowy element oraz indeks, który chce zmienić w wybranej liście.
Potrzeba użycia słownika do przechowywania nazw list jest uzasadniona, ponieważ umożliwia elastyczne odwoływanie się do odpowiedniej listy zależnie od wyboru użytkownika, co ułatwia ewentualne modyfikacje nazw list bez konieczności edycji struktury kodu.

1.3 Algorytm działania w standardzie UML

Poniżej prezentuje diagram w standardzie UML programu. Pierwszy diagram prezentuje schemat działania strukturalnego programu, natomiast drugi prezentuje działanie każdej z użytych funkcji. Do wykonania diagramu został użyty program draw.io.



Rys. 15. diagram programu w standardzie UML



Rys. 16. diagramy funkcji używanych w programie w standardzie UML