**Лабораторна робота №4**

Завдання: Використовуючи теоретичне відомості розробити програму яка на вхід отримує математичний вираз з довільною кількістю операндів, операторів та дужок. В першу чергу сформувати послідовність символів у ЗПН. На другому етапі виконання лабораторної роботи вирахувати результат послідовності, що була сформована, використовуючи алгоритм запису математичного виразу у ЗПН.

Виконання завдання:

1. Спочатку створимо функцію, яка буде отримувати звичайний математичний вираз, а повертатиме вже вираз зворотного польського запису. Цю функцію ми назвемо RPNGenerator() і вона буде приймати параметри expression. Далі треба цю функцію буде побудувати за таким алгоритмом:
2. У випадку, коли є символи для обробки необхідно зчитати символ.
3. Якщо символ є числом, то додаємо його до вихідного рядка.
4. Якщо символ є дужкою, поміщаємо його в стек.
5. Якщо символ є дужкою, що закривається то доки верхнім елементом стека не стане відкриваюча дужка, виштовхуємо елементи зі стека у вихідний рядок. При цьому дужка, що відкриває, видаляється зі стека, але у вихідний рядок не додається. Якщо стек закінчився раніше, ніж ми зустріли дужку, це означає, що у виразі або неправильно поставлений роздільник, або не узгоджені дужки.
6. Якщо символ є бінарною операцією та операція на вершині стеку має більший або такий самий пріоритет, то необхідно “виштовхнути” верхній елемент до вихідного рядка. Помістити операцію в стек.
7. Коли вхідний рядок закінчився, виштовхуємо всі символи зі стека у вихідний рядок.

Знаючи цей алгоритм, реалізуємо функцію.

Створюємо словар операцій і призначаємо як значення пріоритети(найвищий у піднесення до степеня, найменший у додавання, наприклад). За дужками слідкуватимемо в процесі. Далі створюємо масиви stack і output. Далі за допомогою for перебираємо кожне значення в отриманій строці expression і ставимо різні умови та дії для них. Якщо значення виявиться числом, то додати його до масиву output(надалі у вихідний рядок), інакше якщо значення відкрита дужка, то додати її в масив stack(надалі у стек), якщо значення є закрита дужка, то допоки(while) стек не буде пустим і елементом ідучи ззаду не стане відкрита дужка, то елементи із стеку забирати у вихідний рядок. Після цього видалимо із стеку відкриту дужку. Інакше якщо значення є в словарі операції, то допоки(while) стек не буде пустим і поки на вершині стеку знаходиться оператор із пріоритетом, який не менший за пріоритет поточного оператора char, то ми видаляємо цей оператор зі стеку і додаємо його до вихідного рядка. І далі після цього додаємо поточний оператор до стеку. Далі цикл for закінчується за допомогою while вибираємо все зі стеку і вже додаємо до вихідного рядку. І далі з’єднуємо масив output в одну строку і пишемо return ‘ ’.join(output), для того, щоб функція повертала результат – вихідний рядок.

Але є одна проблема. Якщо користувач напише багатоцифрове число, наприклад, 10, то воно сприйметься як 1 та 0, утім так не повинно бути. Тому я імпортував модуль re, модуль для роботи з регулярними виразами. І я роблю таку перевірку якщо вхідний рядок містить щось ще крім символів операцій(^, +, -, \*, /), круглих дужок(), чисел, то повернути, що користувач увів щось неправильно. Інакше за допомогою методу findall з модуля re знаходимо у вхідній строці(expression) числа, які містять одну чи більше цифр чи операції(^, +, -, \*, /), круглі дужки(). Усе те, що знайде метод ми присвоюємо змінній expression, яка буде надалі нашою вхідною строкою, у підсумку за допомогою таких методів я позбувся помилок, що пов’язані з неправильним вводом користувача і позбувся від пробілів у рядку, залишилися в рядку тільки числа, операції та дужки. Код, що написав був до розташовуємо в блоку else. Хоча принципового значення це немає можна просто розташувати після else. Ці перевірки методами перенесемо до нашої другої функції(RPNCalculator()), яка вже буде обчислювати вирази зворотнього польського запису.

1. Створимо функцію RPNCalculator(), яка буде приймати вираз зворотнього польського запису і обчислюватиме його. Це вже буде створити простіше, оскільки нам вже не потрібно відслідковувати як минулій функції пріоритет операцій. Ось такий алгоритм ми будемо використовувати:
2. Проходьте по кожному значенню зворотнього польського запису зліва направо.
3. Якщо символ є операндом, додаємо його до стеку.
4. Якщо символ є оператором, витягаємо два операнди зі стеку, додаємо до них оператор і отриманий вираз знову додаємо до стеку.
5. Продовжуємо цей процес до завершення обробки всього зворотнього польського запису. У результаті після всього в стеку залишиться остаточний результат.

Реалізуємо цей алгоритм. Додамо масив stack. Циклом for перебираємо вхідну строку, далі робимо перевірки і відповідні дії. Якщо значення виявиться числом, то додати його в стек. Інакше видаляємо два числа зі стеку, і перевіряємо значення на те, яку воно має операцію, якщо це +, то додаємо два числа, що ми витягли зі стеку, якщо віднімання, то віднімаємо два числа і так далі аналогічно. Результат дій додаємо назад у стек. І після всього цього в стеку залишиться одне число, так як стек це масив у нас, то отримуємо перший елемент масиву за індексом нуль. І пишемо return stack[0], щоб функція повертала це як результат своєї роботи. Не забудемо викликати ці дві функції, які ми створили, в кінці викликати їх та отримати результат.

1. Тепер необхідно написати тести для цих двох функцій.

Тестувати я буду за допомогою фреймворку для тестування unittest, імпортуємо все необхідне, зробимо початкову структуру для тесту. Імпортуємо наші функції. У функції setUp() визначимо початкові дані для тесту. Це буде чотири масиви, перший – зі звичайними математичними записами, другий – зі зворотними польськими записами, третій – з помилковими вводами користувача і четвертий – із відповідями математичних записів. Перший і другий масив відповідають один одному. Тобто, звичайний математичний вираз дорівнює зворотному польському запису, це для зручності тестування. Далі напишемо дві тест-функції для двох наших функцій. У тест-функції test\_RPNGenerator() створюємо змінну «і» для цикла while та далі допоки «і» менше, ніж довжина масиву звичайних математичних записів, то за допомогою конструкції self.assertEqual(RPNGenerator(self.math\_common\_entries[i]), self.rpn\_entries[i]) ми порівнюємо результат функції у яку передали звичайний запис з зворотним польським записом.

Далі аналогічною конструкцією тестуємо на помилковий ввід два різні записи з масиви помилкового вводу. Для порівняння на першій позиції у нас елемент з масиву помилок, який ми передаємо в функцію RPNGenerator() (і вона має повернути повідомлення), а на другій строка, яка є видастися при помилці. Переходимо до другої тест-функції test\_RPNCalculator(). Тут все аналогічно як в першої текст-функції. Код копіюємо весь з попередньої в цю тест-функцію. Щоправда будуть деякі відмінності, розберемо їх. У циклі при при порівнянні буде порівнюватися вже результат передачі аргумента, який є елементом з масиву зворотніх польських записів, у функцію з масивом результатів. Щодо помилок, то просто передаємо вже не у RPNGenerator(), а у RPNCalculator() у якості аргумента елемент масиву помилок, функція має повернути результат, який ми порівнюємо зі строкою, яка буде свідчить про помилку «You entered something wrong! Please, try enter again.». Це все. Я завершив написання лабораторного завдання та написання юніт-тестів для неї. Час перевірити, чи правильно ми все зробили.

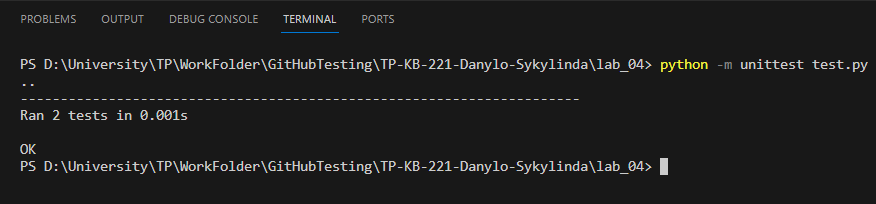


Рис. 1. Результат виконання юніт-тестів.

Як бачимо, всі тести пройшли успішно, отже все виконано правильно.

Код завдання:

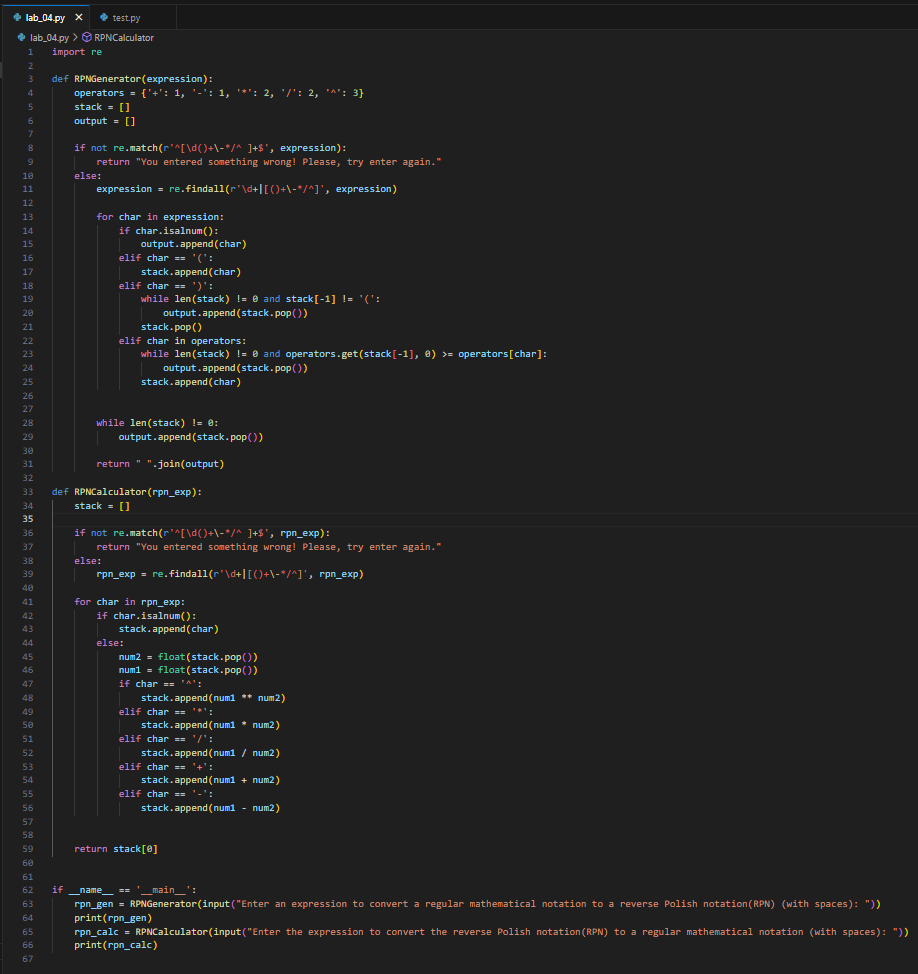


Рис. 2. Код файлу lab\_04.py.



Рис. 3. Код юніт-тестів для коду з файлу lab\_04.py.

GitHub:

Посилання на GitHub на цю папку з файлами лабораторної роботи: <https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/tree/main/lab_04>.

Скріншот коду з GitHub:

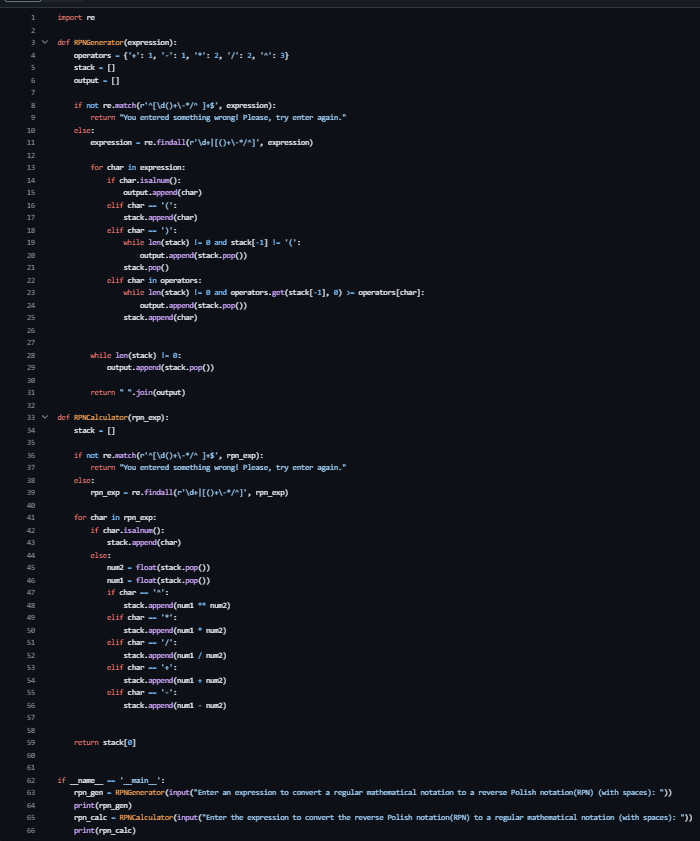


Рис. 4. Код файлу lab\_04.py.



Рис. 5. Код файлу test.py для юніт-тестів для коду файлу lab\_04.py.