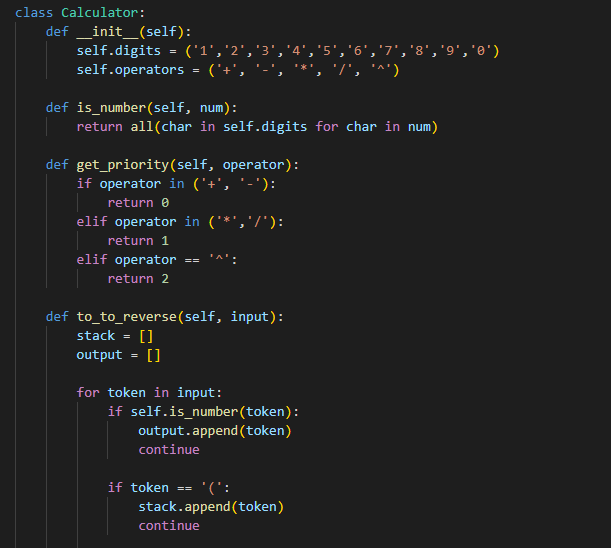
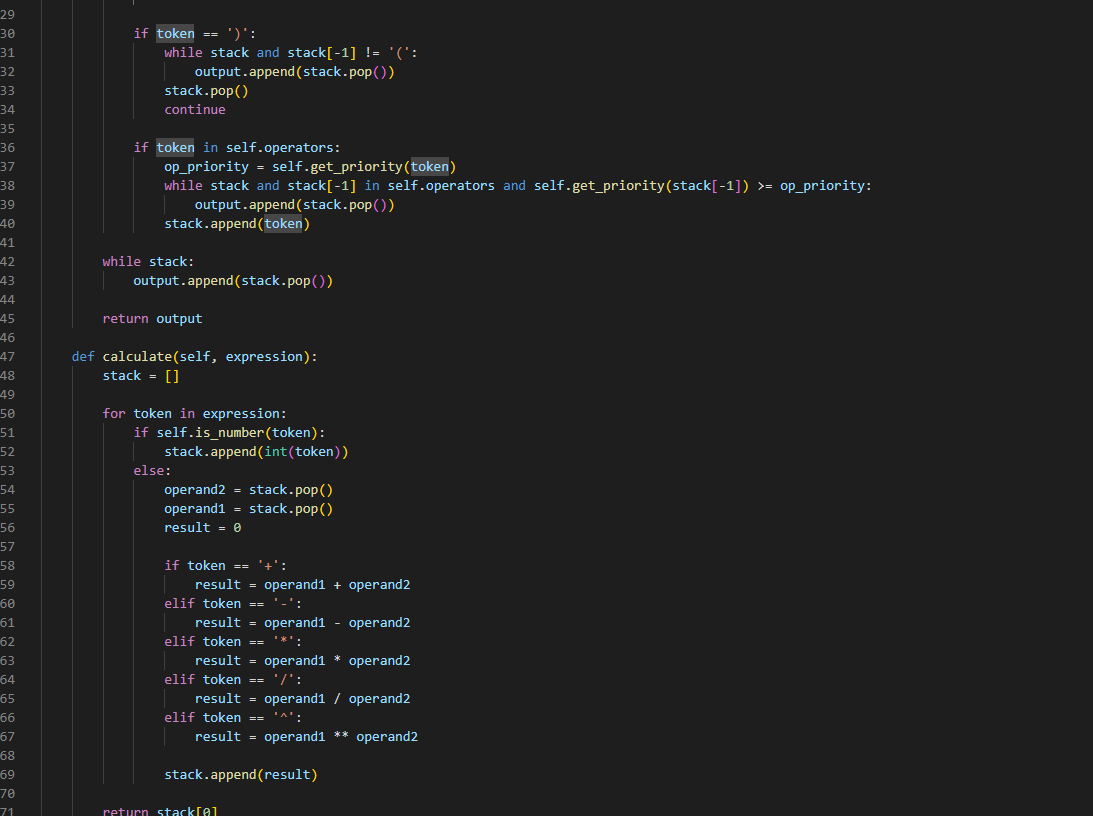
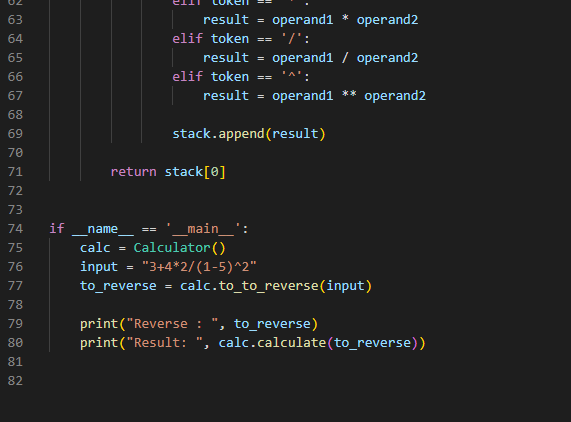
**Лабораторна робота №4**

**Мета роботи:** Використовуючи теоретичне підґрунтя про зворотний польський запис розробити програму на вхід якої подається математичний вираз, що має довільний набір операндів, операторів та дужок, на виході програма обчислює результат математичного виразу.

**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.







Давайте розберемо цей код построчно:

1. class Calculator: - Створюємо новий клас з назвою Calculator.
2. def \_\_init\_\_(self): - Це конструктор класу, який викликається при створенні нового екземпляра класу.
3. self.digits = ('1','2','3','4','5','6','7','8','9','0') - Визначаємо кортеж цифр, які будуть використовуватися для перевірки, чи є токен числом.
4. self.operators = ('+', '-', '\*', '/', '^') - Визначаємо кортеж операторів, які будуть використовуватися в математичних виразах.
5. def is\_number(self, num): - Це метод, який перевіряє, чи є вхідний токен числом.
6. return all(char in self.digits for char in num) - Повертає True, якщо всі символи в токені є цифрами, і False в іншому випадку.
7. def get\_priority(self, operator): - Це метод, який повертає приоритет оператора.
8. if operator in ('+', '-'): return 0 - Якщо оператор є додаванням або відніманням, повертаємо 0.
9. elif operator in ('\*','/'): return 1 - Якщо оператор є множенням або діленням, повертаємо 1.
10. elif operator == '^': return 2 - Якщо оператор є піднесенням до степеня, повертаємо 2.
11. def to\_to\_reverse(self, input): - Це метод, який перетворює вхідний вираз в обернену польську запис.
12. stack = [] - Створюємо стек для зберігання операторів.
13. output = [] - Створюємо список для зберігання вихідного виразу.
14. for token in input: - Проходимо по кожному токену в вхідному виразі.
15. if self.is\_number(token): output.append(token) - Якщо токен є числом, додаємо його до вихідного списку.
16. if token == '(': stack.append(token) - Якщо токен є відкриваючою дужкою, додаємо його до стеку.
17. if token == ')': - Якщо токен є закриваючою дужкою, видаляємо всі оператори зі стеку до відкриваючої дужки.
18. while stack and stack[-1] != '(': output.append(stack.pop()) - Видаляємо оператори зі стеку і додаємо їх до вихідного списку, поки не зустрінемо відкриваючу дужку.
19. stack.pop() - Видаляємо відкриваючу дужку зі стеку.
20. if token in self.operators: - Якщо токен є оператором, додаємо його до стеку, враховуючи приоритет операторів.
21. op\_priority = self.get\_priority(token) - Отримуємо приоритет поточного оператора.
22. while stack and stack[-1] in self.operators and self.get\_priority(stack[-1]) >= op\_priority: output.append(stack.pop()) - Видаляємо оператори зі стеку і додаємо їх до вихідного списку, поки на вершині стеку є оператор з більшим або рівним приоритетом.
23. stack.append(token) - Додаємо поточний оператор до стеку.
24. while stack: output.append(stack.pop()) - Видаляємо всі залишені оператори зі стеку і додаємо їх до вихідного списку.
25. return output - Повертаємо вихідний список як результат.
26. def calculate(self, expression): - Це метод, який обчислює результат виразу в оберненій польській записі.
27. stack = [] - Створюємо стек для зберігання чисел.
28. for token in expression: - Проходимо по кожному токену в виразі.
29. if self.is\_number(token): stack.append(int(token)) - Якщо токен є числом, додаємо його до стеку.
30. else: - Якщо токен є оператором, виконуємо відповідну операцію над двома верхніми числами в стеку.
31. operand2 = stack.pop() - Видаляємо другий операнд зі стеку.
32. operand1 = stack.pop() - Видаляємо перший операнд зі стеку.
33. result = 0 - Ініціалізуємо змінну для зберігання результату.
34. if token == '+': result = operand1 + operand2 - Якщо оператор є додаванням, додаємо операнди.
35. elif token == '-': result = operand1 - operand2 - Якщо оператор є відніманням, віднімаємо операнди.
36. elif token == '\*': result = operand1 \* operand2 - Якщо оператор є множенням, множимо операнди.
37. elif token == '/': result = operand1 / operand2 - Якщо оператор є діленням, ділимо операнди.
38. elif token == '^': result = operand1 \*\* operand2 - Якщо оператор є піднесенням до степеня, підносимо перший операнд до степеня другого операнда.
39. stack.append(result) - Додаємо результат до стеку.
40. return stack[0] - Повертаємо результат як результат обчислення виразу.
41. if \_\_name\_\_ == '\_\_main\_\_': - Це умова, яка виконується, якщо цей файл виконується як основний скрипт.
42. calc = Calculator() - Створюємо новий екземпляр класу Calculator.
43. input = "3+4\*2/(1-5)^2" - Визначаємо вхідний вираз.
44. to\_reverse = calc.to\_to\_reverse(input) - Перетворюємо вхідний вираз в обернену польську запис.
45. print("Reverse : ", to\_reverse) - Виводимо обернену польську

**Висновок :** Ця робота демонструє ефективність використання стеку для обчислення математичних виразів. Ми створили клас Calculator, який може перетворювати вхідний вираз в обернену польську запис і обчислювати результат виразу.Метод to\_reverse\_polish використовує стек для зберігання операторів і враховує їх приоритети, щоб правильно впорядкувати їх в оберненій польській записі. Метод calculate також використовує стек, але в цьому випадку для зберігання чисел і виконання операцій над ними.Цей код є гнучким і легко масштабується. Ми можемо додати нові оператори або змінити приоритети існуючих, просто змінивши відповідні місця в коді. Крім того, він легко інтегрується з іншими програмами, оскільки весь функціонал згрупований в одному класі.В цілому, ця робота демонструє, як можна використовувати стеки для розв’язання складних проблем, таких як обчислення математичних виразів. Завдяки цьому коду, ми можемо краще зрозуміти, як працюють стеки і як вони можуть бути використані в програмуванні.