Лабораторная работа №1. Отчёт

Тема:

Разработка консольных калькуляторов на Python (вариант Medium 2)

Автор:

Попков Максим Александрович, М8О-105БВ-25

Цели работы:

- Закрепить основы Python (ввод/вывод, функции, модули, тесты).
- Освоить базовые техники разбора выражений: токенизация, приоритеты, стек.
- Научиться разделять логику по функциям.
- Практиковать обработку ошибок и написание простых автотестов.

Операции (по приоритетам)

1. Унарные `+` и `-`, право-ассоциативные
2. Возведение в степень `**`, право-ассоциативное
3. `*`, `/`, `//`, `%`, лево-ассоциативные

   (`//`, `%` поддерживаются только для целых чисел,

   неявное умножение не поддерживается)
4. Бинарные `+`, `-`, лево-ассоциативные

Описание алгоритма:

M2 — Шунтирующий двор со скобками.

- **Шаг 1: токенизация операторов и чисел.**

С помощью регулярного выражения строка разбивается на токены: числа, операторы, скобки:

Токен	Описание
NUM	Число (целое или вещественное)
OP	Операторы: +, -, *, **, /, //, %
LP	Открывающаяся скобка (
RP	Закрывающаяся скобка (
WS	Пробелы, знаки табуляции, переносы строк
MIS	Прочие символы, в том числе ., не являющиеся разделителями целой и дробной частей в числах

На этом этапе из исходных данных исключаются пробелы, знаки табуляции и переносы строк `WS`. В случае наличия недопустимых символов `MIS`, например, букв или точек, не являющихся разделителями целой и дробной частей вещественных чисел, выбрасывается ошибка `TokenizeError`.

- **Шаг 2: преобразование инфиксного выражения в RPN с учётом скобок и приоритетов.**
  - **2.1: определение унарных операторов `+` и `-`.** Осуществляется последовательный перебор токенов, в котором при нахождении `+` или `-` проверяется предшествующий токен. В случае, если он отсутствует, либо является операцией или открывающейся скобкой, знак считается унарным. Унарные `-` помечаются как `<NEG>`, а унарные `+` игнорируются и не добавляются в новый список токенов.
  - **2.2: проверка корректности переданного арифметического выражения:**
    - наличие токенов;
    - первый токен не может быть бинарной операцией `OP`, закрывающейся скобкой `RP`;
    - подряд идущие пары токенов не могут быть: `NUM NUM`, `RP NUM`, `OP bOP`, `LP bOP` (`bOP` - бинарная `OP`), `NUM LP`, `RP LP`, `LP RP`, `OP RP`;
    - последний токен не может быть открывающейся скобкой `LP`, операцией `OP`;
    - число открывающихся скобок `LP` должно быть равно числу закрывающихся `RP`.

    При несоответствии выбрасывается ошибка `ParseError`.
  - **2.3: перестановка токенов из инфиксного порядка в RPN (обратную польскую нотацию).** Осуществляется последовательный перебор токенов. Числа `NUM` сразу попадают в результирующий список `rpn`, открывающиеся скобки `LP` в стек операций `stek_op`. Если встречаем операцию `OP`, выталкиваем из `stek_op` в `rpn` операции с таким же приоритетом или выше (для правой ассоциативности исключительно выше), пока не дойдем до неподходящего под это условие оператора `OP` или открывающейся скобки `LP`. После этого добавляем текущую `OP` в `stek_op`. Если встречаем закрывающуюся скобку `RP`, выталкиваем из `stek_op` в `rpn` все операции, пока не дойдем до открывающейся скобки `LP`, при ее отсутствии - ошибка `ParseError`. В конце выталкиваем из `stek_op` оставшиеся операции в `rpn`. Если в `stek_op` осталась открывающаяся скобка `LP` - ошибка `ParseError`.

- **Шаг 3: вычисление RPN стеком.**

Осуществляется последовательный перебор токенов в обратной польской нотации `rpn`. Числа `NUM` размещаются в стек `stek`. Отдельно проработан случай, когда у целых чисел тип данных `float` - переводим в `int`. Если встречаем операцию, унарную применяем к последнему элементу `stek`, бинарные - к двум последним. При этом при любом виде деления (`/`, `//`, `%`) осуществляется проверка, что делитель не равен 0, иначе - ошибка `EvalError`. Для операций `//` и `%` проверяется, что делимое и делитель имеют тип данных `int`, иначе - ошибка `EvalError`. После окончания перебора токенов в `stek` должно остаться одно число - результат вычислений, иначе - ошибка `EvalError`.

Архитектура

Проект разбит на отдельные модули, каждый из которых выполняет одну задачу:

Модуль	Назначение
`tokenizer.py`	Берет арифметическое выражение и разбивает его на токены - числа, операции, скобки
`parser.py`	Определяет унарные минусы и игнорирует унарные плюсы. Проверяет корректность переданного арифметического выражения. Меняет порядок токенов, дано обычное выражение, создает обратную польскую нотацию
`evaluator.py`	Вычисляет арифметическое выражение по записи в обратной польской нотации
`calculator.py`	Вычисляет значение арифметического выражения, с помощью него можно использовать калькулятор как модуль
`constants.py`	Хранит приоритеты и ассоциативность операторов
`errors.py`	Определяет собственные типы ошибок (`TokenizeError`, `ParseError`, `EvalError`).
`main.py`	Точка входа в программу, пользовательский интерфейс

Каждая функция документирована с помощью docstring и аннотаций типов.

Пример работы программы

```
Введите выражение: 3 * (1 + 2)
Результат: 9

Введите выражение: 2**3**2
Результат: 512

Введите выражение: -(3 + 5)
Результат: -8

Введите выражение: 5 // 2
Результат: 2
```

Обработка ошибок

- Недопустимый символ или символ ".", не являющийся разделителем целой и дробной частей вещественного числа - `TokenizeError`.
- Некорректность арифметического выражения (недопустимые токены подряд, некорректные скобочные выражения и т.д.) - `ParseError`.
- Деление на 0 или несоответствие типа данных для операций `//` и `%` - `EvalError`.

Тестирование

Тестирование осуществляется с помощью модуля `pytest`.

Покрыты следующие методы:

- токенизация (`test_tokenizer.py`);

- определение унарных операций (`test_annotate_unary.py`);
- проверка корректности выражения (`test_correctness_check.py`);
- перестановка токенов в RPN (`test_to_rpn.py`);
- вычисление RPN стеком (`test_eval_rpn.py`);
- полное вычисление выражения в инфиксной записи (`test_calc.py`).

Вывод

Разработан калькулятор, вычисляющий значения арифметических выражений в инфиксной записи со скобочными выражениями и унарными операциями. Все цели лабораторной работы выполнены: логика разделена по функциям, которые сгруппированы по модулям; ошибки обрабатываются уникальными классами; для тестирования использованы автотесты.