**Грамматика**

S -> Variable=Expr

Expr -> Sum

Sum -> Sum + Product | Sum - Product | Product | ε

Product -> Product ^ Term | Product \* Term | Product / Term | ε

Term -> (Expr) | Number | Variable

----------------------------------------

Number – число типа int или float

Variable – переменная

**Переход КСГ в парсер**

**Об операндах**

number = pyparsing\_common.number

Так задаются числа, участвующие в выражениях (Number в грамматике). pyparsing\_common.number приводит полученные символы к числу.

variable = OneOrMore(Word(alphas))

Так задаются имена переменных (Variable в грамматике). OneOrMore(Word(alphas)) считает именем переменной от одного и более символов из набора латинских букв обоих регистров.

operand = number | variable

Так указываются операнды выражения. В этом простом случае это или переменная, или число.

**Об операторах**

expop = Literal('^')

Так задается операция возведения в степень с помощью символа ^ и класса парсера. (В описании грамматики такой же символ)

multop = oneOf('\* /')

Так задается операция деления или умножения с помощью символа \* или / и класса парсера. (В описании грамматики такие же символы для этих действий)

plusop = oneOf('+ -')

Так задается операция сложения или вычитания с помощью символа + или - и класса парсера. (В описании грамматики такие же символы для этих действий)

**О выражениях**

expr = variable + '=' + infixNotation(

operand,

[(expop, 2, opAssoc.RIGHT),

(multop, 2, opAssoc.LEFT),

(plusop, 2, opAssoc.LEFT),]

)

Так задаются все выражения из файла std.in (это первая строка из описания грамматики). Выражение начинается с переменной, потом к нему добавляется знак равенства и остальная часть выражения в виде функции парсера infixNotation. Функция в качестве аргументов берет набор операнды – operand и список возможных операций. Операции расположены по их приоритету. В каждом элементе списка 3 параметра: оператор, объявленный выше, число операндов, для которых применяется оператор, тип ассоциативности – левая или правая. Таким образом любая строка из файла std.in будет представлена в виде выражения expr.

var, equal, statement = expr.parseString(line, parseAll=True)

С помощью метода parseString каждое выражение разделяется на переменную var, знак равенства equal и само условие выражения statement.

variables[var] = transform(statement)

Далее функция transform рекурсивно вычисляет значение выражения (по ходу используется функция acc, которая берет пару операндов и оператор и возвращает результат операции), пока не получится ответ. Эти действия эквивалентны строкам 2-5 грамматики. Полученный ответ записывается в словарь переменных variables, чтобы в последующих выражениях можно было использовать предыдущие операнды.

После обработки строки печатается ответ выражения, либо ошибка, обнаруженная парсером при разборе. (либо печатается ошибка, найденная при вычислениях)

**Запуск**

Строки с выражениями должны быть написаны в файле с именем ‘std.in’.

Файл интерпретатора называется hw\_10.py

Файл интерпретатора должен находиться в той же директории, что и файл с выражениями.

Для запуска выполните команду: python hw\_10.py

Вывод пройдет прямо в консоль, если в строке была ошибка, то описание ошибки будет выведено вместо выражения.

**Поддерживаемые операции**

+, -, /, \*, ^

Операции можно группировать с помощью обычных скобок «()».

Деление на ноль запрещено.

Переменную можно задавать повторно.

В выражениях можно использовать ранее объявленные переменные.

Можно работать с целыми и вещественными числами

**Синтаксис**

Объявление или переопределение переменной происходит с помощью знака «=», например, «a =» или «a = 3 + 5».

Для имен переменных можно использовать любые символы из набора 'abcdefghijklmnoprstuvwxyz' и их строчные варианты.

Каждое новое выражение должно быть написано на новой строке.

Для вывода значения ранее созданной переменной необходимо выполнить выражение, в котором переменная равна сама себе, например, «k = k».

Пример работы

|  |  |
| --- | --- |
| Строки в файле | Вывод программы |
| a = 2.5  b = ((3))  K = 6 - 7  as = 3 \* 4 + 5 \* 6.8  rtd = 2 / 1  o = 0 + 9  c = (2 \* (5 - 3))  d = b - a  r = 18 / 6 - 5 \* (o / ( b - 2)) + 49 - (k + 1)  e =  dfgh = ik  чс = 5 + 5  n = 8  n = a  n = a  n = b  fg = 6+7  yui = 3 / 0  adh = -  dfgh = 2 ^ o  t = ((3+4)  sdf = 18 / 6 - 5 \* (o / ( b - (2)) + 49 - (k + 1)  ert :=7  step = 2 ^ 3 \* 5 ^ 2  hj = 2 -5 ^2 | a = 2.5  b = 3  K = -1  as = 46.0  rtd = 2.0  o = 9  c = 4  d = 0.5  Переменная k неопределена  Expected {+ | - term | \* | / term} (at char 5), (line:2, col:1)  Переменная ik неопределена  Expected W:(ABCD...) (at char 0), (line:1, col:1)  n = 8  n = 2.5  n = 2.5  n = 3  Expected W:(ABCD...) (at char 1), (line:2, col:1)  fg = 13  Деление на ноль  Expected {+ | - term | \* | / term} (at char 6), (line:1, col:7)  dfgh = 512  Expected {+ | - term | \* | / term} (at char 11), (line:2, col:1)  Expected end of text (at char 17), (line:1, col:18)  Expected "=" (at char 4), (line:1, col:5)  step = 200  hj = -23 |