МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
Факультет прикладної математики та інформатики
Кафедра програмування

Лабораторна робота №12

Інтерпретація за формулами

Виконала
студентка групи ПМО-41
Кравець Ольга

2024

Хід роботи

У цій лабораторній роботі я використала код з файлу arithexpr.py як основу і зробила в ньому потрібні зміни.

Коментарі в коді написані англійською мовою, щоб їх було простіше відрізнити від тих коментарів які там вже були.

Нові лексеми:

Оголошую в конструкторі

```
# Added new operators
divideWhole = 9
remainder = 10
sin = 11
pow = 12
```

В методі scanner додала їх до списку

```
    def scanner(self):  # сканувати формулу і поділити на лексеми
            elif self.text[self.i] == '-':
                self.leks.append((self.subtract, '-'))
            elif self.text[self.i] == '*':
                # Added check for raising to power
                if self.text[self.i + 1] == '*':
                    self.i += 1
                    self.leks.append((self.pow, '**'))
                else:
                    self.leks.append((self.multiply, '*'))
            elif self.text[self.i] == '/':
                # Added check for whole division
                if self.text[self.i + 1] == '/':
                    self.i += 1
                    self.leks.append((self.divideWhole, '//'))
                else:
                    self.leks.append((self.divide, '/'))
            # Added check for remainder
            elif self.text[self.i] == '%':
                self.leks.append((self.remainder, '%'))
            elif self.text[self.i].isdigit():
                self.onenumber()
                self.i -= 1
            else:
                # Detect function
                if self.funcname() is None:
                    return False  # недопустима літера
                self.i -= 1
            self.i += 1
        self.leks.append((self.empty, '#'))  # обмежувач списку лексем
        return True
```

Метод funcname для знаходження викликів функцій:

```
# Function to get function name from string
def funcname(self):
    name = ''
    while self.i < len(self.text):
        name += self.text[self.i]
        self.i += 1
        if name == 'sin':
            self.leks.append((self.sin, 'sin'))
            return True
    return None
```

Нові бінарні операції в методі term:

```
def term(self):
    z = self.factor()  # найперший множник: правило term ::= factor
    while self.leks[self.k][0] == self.multiply \
            or self.leks[self.k][0] == self.divide \
            or self.leks[self.k][0] == self.remainder \
            or self.leks[self.k][0] == self.divideWhole:
        # наступні множники: правило  term ::=  ( ( "*" | "/" )  factor ) *
        opr = self.leks[self.k][0]  # запам'ятати операцію
        self.GetNextToken()  # перейти до наступної лексеми
        # Added conditions for remainder and whole divisions
        if opr == self.multiply:
            z = z * self.factor()
        elif opr == self.divide:
            z = z / self.factor()
        elif opr == self.remainder:
            z = z % self.factor()
        else:
            z = z // self.factor()
    return z
```

Змінила метод factor:

Додала перевірку наявності унарних операцій на початку:

```
# Added rule factor ::= [(+ | -)]
is_negative = False
if self.leks[self.k][0] == self.add:
    self.GetNextToken()
elif self.leks[self.k][0] == self.subtract:
    self.GetNextToken()
    is_negative = True
```

Додала правило виклику функції:

```
# Added rule factor ::= function
elif self.leks[self.k][0] == self.sin:
    sub_result = self.functions()
else:
    return None
```

Якщо був унарний мінус записуємо це в підрезультат:

```
if is_negative:
    sub_result = -sub_result
```

Якщо наступна лексема піднесення до степеня, то робимо це в кінці методу factor (за правилами алгебри з пункту 5 файлу з завданням):

```
# Added rule factor ::= [( "**" factor ) *]
if self.leks[self.k][0] == self.pow:
    self.GetNextToken()
    return sub_result ** self.factor()
else:
    return sub_result
```

Тепер метод factor має такий вигляд:

```
def factor(self):
    # Added rule factor ::= [(+ | -)]
    is_negative = False
    if self.leks[self.k][0] == self.add:
        self.GetNextToken()
    elif self.leks[self.k][0] == self.subtract:
        self.GetNextToken()
        is_negative = True

    if self.leks[self.k][0] == self.number:  # правило  factor ::= number
        self.GetNextToken()  # перейти до наступної лексеми
        sub_result = self.leks[self.k - 1][1]  # повернути число попередньої лексеми
    elif self.leks[self.k][0] == self.openbracket:
        # правило  factor ::=  "(" arith_expr ")"
        self.GetNextToken()  # перейти до наступної лексеми
        ex = self.arithexpr()  # частина виразу в дужках
        if self.leks[self.k][0] == self.closebracket:
            self.GetNextToken()
        else:
            raise errorexpr  # ? немає закриваючої дужки
        sub_result = ex
    # Added rule factor ::= function
    elif self.leks[self.k][0] == self.sin:
        sub_result = self.functions()
    else:
        return None

    if is_negative:
        sub_result = -sub_result

    # Added rule factor ::= [( "**" factor ) *]
    if self.leks[self.k][0] == self.pow:
        self.GetNextToken()
        return sub_result ** self.factor()
    else:
        return sub_result
```

Метод виклику функцій:

```
# Added function with rule function ::= "sin(" arith_expr ")"
def functions(self):
    opr = self.leks[self.k][0]
    if opr == self.sin:
        self.GetNextToken()
        if self.leks[self.k][0] == self.openbracket:
            self.GetNextToken()
            ex = self.arithexpr()
            if self.leks[self.k][0] == self.closebracket:
                self.GetNextToken()
            else:
                raise errorexpr
            if opr == self.sin:
                return math.sin(ex)
    return None
```

Зміни в методі onenumber для сприйняття не тільки цілих чисел:

```
# Added method to detect if number is valid
def isFloat(self, value):
    try:
        float(value)
        return True
    except ValueError:
        return False
```

```
def onenumber(self):  # читати літери числа - правило  number::= cipher  cipher *
    num = ""
    while self.i < len(self.text):
        num += self.text[self.i]

        # Additional checks for scientific notation
        if self.text[self.i] == 'E':
            num += self.text[self.i + 1]
            self.i += 1

            if self.text[self.i] == '+' or self.text[self.i] == '-':
                self.i += 1
                num += self.text[self.i]

        self.i += 1

        if not self.isFloat(num):
            num = num[:-1]
            self.i -= 1
            break

    if len(num) > 0:
        self.leks.append((self.number, float(num)))     # Changed into to float
    else:
        return None
```

Тестування:

Для тестування був створений список кортежів, який має вираз та результат, який повинен вийти в результаті. Якщо ми не отримаємо цей результат, у консолі це буде виведено.

Остача:

```
if __name__ == "__main__":
    formulas = [
        # Testing remainder
        ("17 % 4", 1),
        ("17 % 5", 2),
        ("17 % 6", 5),
        ("17 % 17", 0),
        ("17 % 18", 17),
```

```
TERMINAL
Expression:
17%4
Result: 1.0
Expression:
17%5
Result: 2.0
Expression:
17%6
Result: 5.0
Expression:
17%17
Result: 0.0
Expression:
17%18
```

Ділення на ціло:

```
# Testing whole division
("17 // 4", 4),
("17 // 5", 3),
("17 // 6", 2),
("17 // 17", 1),
("17 // 18", 0),
```

```
TERMINAL
Expression:
17//4
Result: 4.0
Expression:
17//5
Result: 3.0
Expression:
17//6
Result: 2.0
Expression:
17//17
Result: 1.0
Expression:
17//18
Result: 0.0
```

Піднесення до степеня:

```
# Testing raising to power
("4 ** 0.5", 2),
("(2 + 2) ** (4 % 2)", 1),
("(2 + 2) ** (5 % 3)", 16),
("5 * 3 ** 2", 45),
```

```
TERMINAL
Expression:
4**0.5
Result: 2.0
Expression:
(2+2)**(4%2)
Result: 1.0
Expression:
(2+2)**(5%3)
Result: 16.0
Expression:
5*3**2
Result: 45.0
```

Функції:

```
# Testing sin (we can use 3.14 as Pi because they are rounded when compared to test)
("sin(3.14)", 0),
("sin(3.14 / 6)", 0.5),
("sin(3.14 / 3)", 3 ** 0.5 / 2),
```

```
TERMINAL
Expression:
sin(3.14)
Result: 0.0015926529164868282
Expression:
sin(3.14/6)
Result: 0.4997701026431024
Expression:
sin(3.14/3)
Result: 0.8657598394923444
```

Унарні операції:

```
# Testing unary operators + -
("-3 * 4", -12),
("3 * -4", -12),
("-(-3 * -4) / 2", -6),
("5--2", 7),
("3-+3", 0),
("3+-3", 0),
```

```
TERMINAL
Expression:
3*-4
Result: -12.0
Expression:
-(-3*-4)/2
Result: -6.0
Expression:
5--2
Result: 7.0
Expression:
3-+3
Result: 0.0
Expression:
3+-3
Result: 0.0
```

Використання дробових чисел:

```
# Testing using floats
("1.45 * 2", 2.9),
("1.33 * 3", 3.99),
```

```
TERMINAL
Expression:
1.45*2
Result: 2.9
Expression:
1.33*3
Result: 3.99
```

Використання експоненційного запису:

```
# Testing using scientific notation
("1E1", 10),
("1E2", 100),
("10E2", 1000),
("1E-3 * 1E+3", 1),
("1E4 * 1E3", 1E7),
```

```
TERMINAL
Expression:
1E1
Result: 10.0
Expression:
1E2
Result: 100.0
Expression:
10E2
Result: 1000.0
Expression:
1E-3*1E+3
Result: 1.0
Expression:
1E4*1E3
Result: 10000000.0
```