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

Лабораторна робота №11

Інтерпретація без рангів

Виконала
студентка групи ПМО-41
Кравець Ольга

2024

Хід роботи

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

Спочатку перенесла оператори як поле класу.

```
# перший перегляд - сканування формули і будова списку лексем
# другий перегляд - обчислення формули
import math

class SimpleInterpret:

    def __init__(self, text):  # конструктор
        # Додала оператори як поля класу
        self.unary = ['r', 'p', 's']
        self.binary = ['+', '-', '*', '/', '%', 'm']

        self.operators = self.unary + self.binary
```

Відповідно в інших місцях, де були перевірки на приналежність до операторів, замінила на ці зміні.

```
def onesign(self):  # читати знак операції - правило  operator::= "+" | "-" | "*" | "/"
    if self.text[self.i] in self.operators:  # Тут зміни
```

У ці же змінні додала нові оператори - % та m для остачі та меншого з результату та запропонованого числа. Вони були запропоновані в завданні. Також додала унарні оператори r, p, s для квадратного кореня, множення на π та функції синусу, відповідно.

Для їх роботи в методі scanner додала умову, щоб для унарних операторів не вимагалось число після нього.

```
if sign not in self.unary:   # Додала цей рядок для унарних операцій
    n = self.onenumber()  # наступна позиція - число
    if n != None:
        self.leks.append(n)
    else:
        return None  # помилка в числі
```

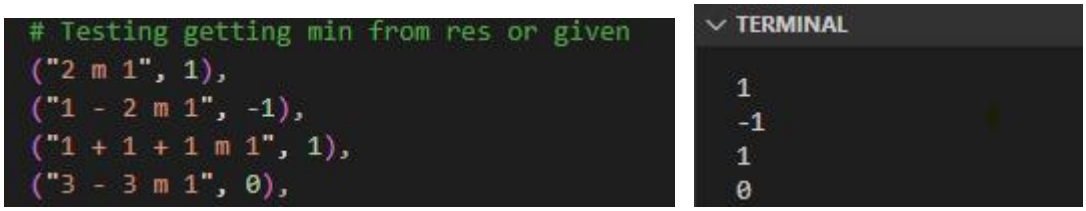
У методі calc додала умову для того, щоб ми не брали наступний елемент списку, якщо оператор унарний.

```
if oper in self.unary: # Додала цей рядок для унарних операцій
    res = res ** 0.5 if oper == 'r' else res * math.pi if oper == 'p' \
        else round(math.sin(res), 4)
```

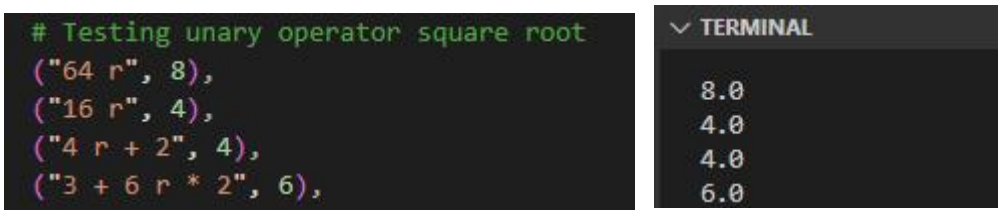
Для тестування нових операторів перевірила результат обчислень зі справжніми даними: якщо вони неправдиві, програма про це каже. Кожен тестовий сценарій є елементом списку формул, разом з результатом цього сценарію. Результати тестуванняя:

Остача:

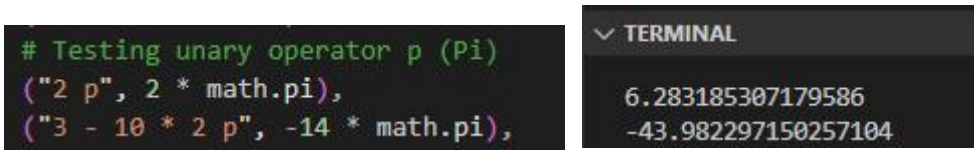
Мінімальне з результату та наступного:



Квадратний корінь :



Множення на π:



Синус (табличні значення):

```
# Testing unary operator s (sin)
("0 s", 0),
("1 / 6 p s", 0.5),
("1 / 4 p s", round(1 / (2 ** 0.5), 4)),
("1 / 3 p s", round((3 ** 0.5) / 2, 4)),
("1 / 2 p s", 1),
("1 p s", 0),
("2 p s", 0),
```

```
TERMINAL

0.0
0.5
0.7071
0.866
1.0
0.0
-0.0
```

За результатами тестування видно, що нові додані бінарні й унарні оператори в переписаному коді працюють добре.

Для підтримки двійкової та шістнадцяткової систем числення додала потрібні поля в клас.

```
# Додала ці поля, щоб відслідковувати в якій системі числення робити операції
self.is_hex = False
self.is_bin = False
```

Також додала обгортки, щоб розрізняти в якій системі числення потрібно вести обрахунки.

```
# Додала нові обгортки для кожної системи обчислення
def dec_calc(self):
    self.is_bin = False
    self.is_hex = False

    return self.calc()

def bin_calc(self):
    self.is_bin = True
    self.is_hex = False

    result = self.calc()
    if result[0]:
        return True, bin(result[1])
    return result

def hex_calc(self):
    self.is_bin = False
    self.is_hex = True

    result = self.calc()
    if result[0]:
        return True, hex(result[1])
    return result
```

Обгортки встановлюють потрібні поля і конвертують отриманий результат в потрібну систему числення.

Переписала повністю метод отримання числа, щоб всі вхідні дані були тільки в заданій системі числення.

```
def onenumber(self):  # читати літери числа - правило  number::= cipher  cipher *
    # Цей метод майже повністю переписаний для підтримки двійкової та шістнадцяткової систем числення
    if self.is_bin:
        num = "0b"
    elif self.is_hex:
        num = "0x"
    else:
        num = ""
    while self.i < len(self.text) and self.text[self.i] not in self.operators:  # Тут зміни
        if self.is_bin and self.text[self.i] != '0' and self.text[self.i] != '1':
            return None
        if self.is_hex and not self.text[self.i].isdigit() and \
                self.text[self.i] != 'A' and self.text[self.i] != 'B' and self.text[self.i] != 'C' and \
                self.text[self.i] != 'D' and self.text[self.i] != 'E' and self.text[self.i] != 'F':
            return None
        if not self.is_bin and not self.is_hex and not self.text[self.i].isdigit():
            return None
        num += self.text[self.i]
        self.i += 1
    if len(num) > 0:
        return num
    else:
        return None
```

У методі для обрахунку додала умови для правильної конвертації в десяткову систему числення для обрахунків. Після отримання результату обгортка перетворить його з 10 системи, в яких були обрахунки, на потрібну перед тим як повернути користувачу.

Тестування двійкової системи:

```
binary_calcs = [
    ("101 + 11", 0b1000),
    ("101 * 10", 0b1010),
    ("4 * 10", 0),    # Error here
    ("01 + 01 + 01", 0b11),
]
```

```
TERMINAL

0b1000
0b1010
Error symbol: 4
0b11
```

Як видно з тестування, програма правильно відкинула 3 формулу, оскільки число 4 не є в двійковій системі.

Тестування шістнадцядкової системи:

```
hex_calcs = [
    ("AB % 1", 0),  # AB = 171
    ("AB % 2", 1),
    ("AB % 3", 0),
    ("AB % 4", 3),
    ("AB % 5", 1),
    ("AB % AB", 0),
    ("AB % AC", 0xAB),
    ("AB % AC m 10 - F", 0x1),
    ("AB % AC m 10 - G", 0),    # Error here
]
```

```
TERMINAL

0x0
0x1
0x0
0x3
0x1
0x0
0xab
0x1
Error symbol: G
```

Як видно з тестування, усі обчислення працюють правильно і правильно відкинутий останній приклад, оскільки G виходить за межі системи.