Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент перспективной инженерии

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
дисциплины «Объектно-ориентированное программирование»

Вариант №2

Выполнила:
Беседина Инга Олеговна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Проверил:
Воронкин Р. А., канд. технических
наук, доцент, доцент департамента
цифровых, робототехнических систем
и электроники института
перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Элементы объектно-ориентированного программирования в языке Python

Цель: Приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

Ход работы

Пример 1:** *Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где a — числитель, b — знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы операции:*

- сложения add, $(a, b) + (c, d) = (ad + be, bd)$;
- вычитания sub, $(a, b) - (c, d) = (ad - be, bd)$;
- умножения mul, $(a, b) \times (c, d) = (ac, bd)$;
- деления div, $(a, b)/(c, d) = (ad, be)$;
- сравнения equal, greate, less.

Должна быть реализована приватная функция сокращения дроби reduce, которая обязательно вызывается при выполнении арифметических операций.

Программа для решения поставленной задачи:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


class Rational:

    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)

        if b == 0:
            raise ValueError()

        self.__numerator = abs(a)
        self.__denominator = abs(b)

        self.__reduce()


    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a, b):
            if a == 0:
                return b
            elif b == 0:
                return a
            elif a >= b:
```

```
            return gcd(a % b, b)
        else:
            return gcd(a, b % a)

    c = gcd(self.__numerator, self.__denominator)

    self.__numerator //= c
    self.__denominator //= c


@property
def numerator(self):
    return self.__numerator


@property
def denominator(self):
    return self.__denominator


def read(self, prompt=None):
    line = input() if prompt is None else input(prompt)
    parts = list(map(int, line.split('/', maxsplit=1)))

    if parts[1] == 0:
        raise ValueError()

    self.__numerator = abs(parts[0])
    self.__denominator = abs(parts[1])

    self.__reduce()


def display(self):
    print(f"{self.__numerator}/{self.__denominator}")


def add(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator + \
        self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator

        return Rational(a, b)
    else:
        raise ValueError


def sub(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator - \
```

```
            self.denominator * rhs.numerator
            b = self.denominator * rhs.denominator

            return Rational(a, b)
        else:
            raise ValueError


    def mul(self, rhs):
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.numerator
            b = self.denominator * rhs.denominator

            return Rational(a, b)
        else:
            raise ValueError()


    def div(self, rhs):
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator
            b = self.denominator * rhs.numerator

            return Rational(a, b)
        else:
            raise ValueError()


    def equals(self, rhs):
        if isinstance(rhs, Rational):
            return (self.numerator == rhs.numerator) and \
                (self.denominator == rhs.denominator)
        else:
            return False


    def greater(self, rhs):
        if isinstance(rhs, Rational):
            v1 = self.numerator / self.denominator
            v2 = rhs.numerator / rhs.denominator

            return v1 > v2
        else:
            return False


    def less(self, rhs):
        if isinstance(rhs, Rational):
            v1 = self.numerator / self.denominator
            v2 = rhs.numerator / rhs.denominator
```

```
            return v1 < v2
        else:
            return False



if __name__ == '__main__':
    r1 = Rational(3, 4)
    r1.display()

    r2 = Rational()
    r2.read("Введите обыкновенную дробь: ")
    r2.display()

    r3 = r2.add(r1)
    r3.display()

    r4 = r2.sub(r1)
    r4.display()

    r5 = r2.mul(r1)
    r5.display()

    r6 = r2.div(r1)
    r6.display()
```

```
3/4
Введите обыкновенную дробь: 5/6
5/6
19/12
1/12
5/8
10/9
```

Рисунок 1. Результат работы программы

Индивидуальные задания:

Задание 1

Парой называется класс с двумя полями, которые обычно имеют имена *first* и *second*. Требуется реализовать тип данных с помощью такого класса. Во всех заданиях обязательно должны присутствовать:

- метод инициализации `__init__`; метод должен контролировать значения аргументов на корректность;
- ввод с клавиатуры `read`;
- вывод на экран `display`.

Реализовать внешнюю функцию с именем `make_тип()`, где `тип` — тип реализуемой структуры. Функция должна получать в качестве аргументов значения для полей структуры и возвращать структуру требуемого типа. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.

2. Поле first — дробное число; поле second — дробное число, показатель степени. Реализовать метод `power()` — возведение числа first в с тепень second. Метод должен правильно работать при любых допустимых значениях first и second.

Программа для решения поставленной задачи:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import math


def make_exponentiation(a, b):
    if isinstance(a, float) and isinstance(b, float):
        struct = Exponentiation(a, b)

        return struct
    else:
        raise ValueError


class Exponentiation:

    def __init__(self, first=0.0, second=0.0):
        first = float(first)
        second = float(second)

        self.__number = first
        self.__exponent = second

        if first < 0:
            raise ValueError

    @property
    def number(self):
        return self.__number

    @property
    def exponent(self):
        return self.__exponent

    def read(self):
        line = input('Введите: ')
        parts = list(line.split('^', maxsplit=1))

        if "," in line:
            raise ValueError
        elif "/" in line:
            num_exp = []
            for part in parts:
                temp = list(map(int,part.split('/', maxsplit=1)))
```

```
                num_exp.append(temp[0]/temp[1])

            self.__number = num_exp[0]
            self.__exponent = num_exp[1]
        else:
            self.__number = float(parts[0])
            self.__exponent = float(parts[1])

    def display(self):
        print(f"{self.__number}^{self.__exponent}")

    def power(self):
        return math.pow(self.number, self.exponent)


if __name__ == '__main__':
    e1 = make_exponentiation(1/2, 1/5)
    e1.display()

    e1.read()
    e1.display()
    print(e1.power())

    e2 = make_exponentiation(0.5, 0.2)
    e2.display()
    print(e2.power())
```

```
0.5^0.2
Введите: 1/4^-1/3
0.25^-0.3333333333333333
1.5874010519681994
0.5^0.2
0.8705505632961241
```

Рисунок 2. Результат работы программы

Задание 2

Составить программу с использованием классов и объектов для решения задачи. Во всех заданиях, помимо указанных в задании операций, обязательно должны быть реализованы следующие методы:

- метод инициализации `__init__`;
- ввод с клавиатуры `read`;
- вывод на экран `display`.

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ = '__main__':` добавить код, демонстрирующий возможности разработанного класса.

2. Создать класс ModelWindow для работы с моделями экранных окон. В качестве полей задаются: заголовок окна, координаты левого верхнего угла, размер по горизонтали, размер по вертикали, цвет окна, состояние «видимое/невидимое», состояние «с рамкой/без рамки». Координаты и размеры указываются в целых числах. Реализовать операции: передвижение окна по горизонтали,по вертикали; изменение высоты и/или ширины окна изменение цвета; изменение состояния, опрос состояния. Операции передвижения и изменения размера должны осуществлять проверку на пересечение границ экрана. Функция вывода на экран должна индуцировать состояние полей объекта.

Программа для решения поставленной задачи:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


WIN_WIDTH = 1920
WIN_HEIGHT = 1080


class ModelWindow():
    fields = [
    ("Заголовок: ", str),
    ("Цвет: ", str),
    ("Координата по x: ", int),
    ("Координата по y: ", int),
    ("Ширина: ", int),
    ("Высота: ", int),
    ("Видимость (yes/no): ", str),
    ("Рамка (yes/no): ", str)
    ]

    def __init__(self,
                 title="New window", color="white",
                 x=0, y=0,
                 width=500, height=500,
                 vis="yes", border="yes"):
        self.__title = title
        self.__color = color
        self.__coord_x = x
        self.__coord_y = y
        self.__width = width
        self.__height = height
        self.__visibility = vis
        self.__border = border

    def move_x(self, x=0):
        if self.__coord_x + self.__width + x <= WIN_WIDTH and \
        self.__coord_x + x >= 0:
            self.__coord_x += x
        else:
```

```
            print("Окно пересекает границы экрана")

    def move_y(self, y=0):
        if self.__coord_y + self.__height + y <= WIN_HEIGHT and \
        self.__coord_y + y >= 0:
            self.__coord_y += y
        else:
            print("Окно пересекает границы экрана")

    def change_size(self, w=0, h=0):
        if self.__width + w >= 0 and self.__height + h >= 0:
            if self.__coord_x + self.__width + w <= WIN_WIDTH and \
                self.__coord_y + self.__height + h <= WIN_HEIGHT:
                self.__width += w
                self.__height += h
            else:
                print("Окно пересекает границы экрана")
        else:
            print("Ширина/высота не может быть отрицательной")

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, color):
        self.__color = color

    @property
    def visibility(self):
        return self.__visibility

    @visibility.setter
    def visability(self, condition):
        self.__visibility = condition

    @property
    def border(self):
        return self.__border

    @border.setter
    def border(self, condition):
        self.__border = condition

    def read(self):
        values = []
        for field_name, field_type in ModelWindow.fields:
            user_input = input(f"Введите {field_name}")
            if not user_input.isdigit() and field_type == str:
                values.append(field_type(user_input))
            elif user_input.isdigit() and field_type == int:
```

```
                values.append(field_type(user_input))
            else:
                raise TypeError

        i = 0
        for key in self.__dict__.keys():
            object.__setattr__(self, key, values[i])
            i += 1

    def display(self):
        print('\n')
        i = 0
        for value in self.__dict__.values():
            print(f"{ModelWindow.fields[i][0]}{value}")
            i += 1


if __name__ == '__main__':
    w1 = ModelWindow("My window", "blue", 100, 100, 1280, 720)
    w1.display()

    w1.color = "green"
    w1.visability = "no"
    w1.border = "no"
    print(f"Цвет:{w1.color}, видимость:{w1.visibility}, рамка:{w1.border}\n")

    w = ModelWindow()
    w.read()
    w.display()

    w.change_size(120, -50)
    w.move_x(-240)
    w.move_y(60)
    w.display()
```

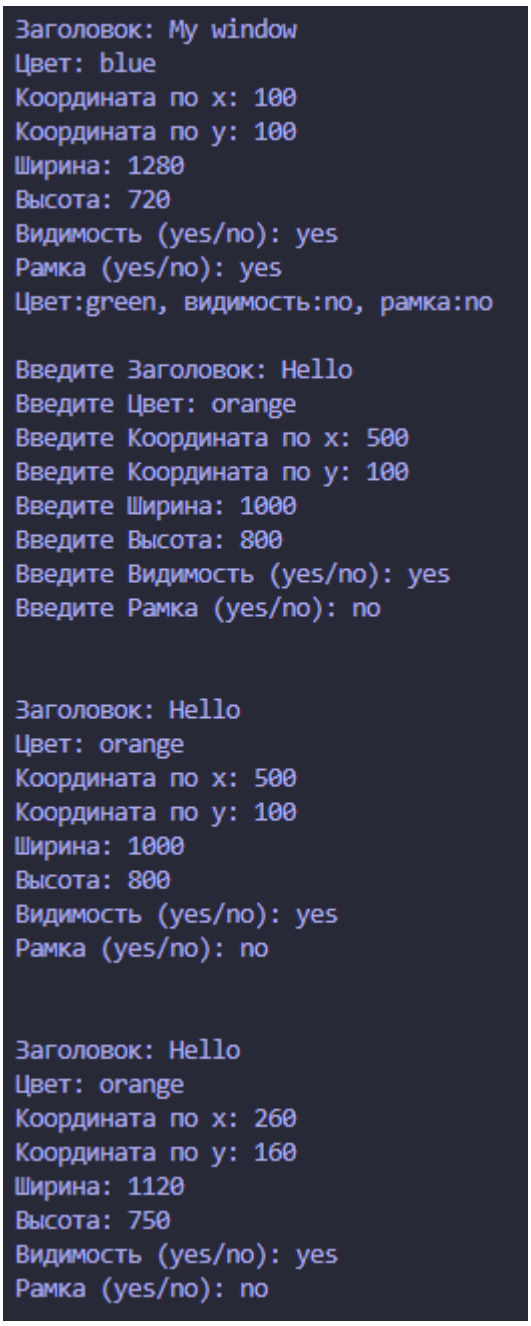

Рисунок 3. Результат работы программы

Контрольные вопросы:

1. Классы объявляются с помощью ключевого слова class и имени класса

2. Атрибуты класса - это атрибуты, которые определены внутри класса, но вне каких-либо методов, и общие для всех экземпляров класса, а атрибуты экземпляра определяются в методах и хранят информацию, специфичную для экземпляра

3. Методы определяют функциональность объектов, принадлежащих конкретному классу

4. Метод __init__ указывает, какие атрибуты будут у экземпляров класса

5. Аргумент self представляет конкретный экземпляр класса и позволяет получить доступ к его атрибутам и методам

6. Для того чтобы добавить атрибуты в класс можно записать их в самом классе с самого начала или создать переменную вне класса, используя точечную нотацию: Car.model = “Lada”

7. В Python нет возможностей для управления доступом к методам и атрибутам. При этом есть соглашение, что метод или атрибут, который начинается с нижнего подчеркивания, является скрытым, и снаружи класса трогать его не нужно (хотя сделать это можно). Если же атрибут или метод начинается с двух подчеркиваний, то тут обратиться к нему напрямую не получиться (простым образом). Хорошим тоном считается, что для чтения/изменения какого-то атрибута должны использоваться специальные методы, которые называются getter/setter

8. Функция isinstance() проверяет, является ли объект экземпляром указанного класса или его подкласса

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python