

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

**Объектно-ориентированное программирование**

**Отчет по лабораторной работе №4.1**

Элементы объектно-ориентированного программирования в языке Python.

Выполнил студент группы

ИВТ-б-о-21-1

Костукайло К.Н. «   » \_\_\_\_\_ 20\_\_ г.

Подпись студента \_\_\_\_\_

Работа защищена «   » \_\_\_\_\_ 20\_\_ г.

Проверил доцент

Кафедры инфокоммуникаций, старший  
преподаватель

Воронкин Р.А.

\_\_\_\_\_  
(подпись)

Ставрополь 2023

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

**Проработка примера.**

**Код программы:**

```
#!/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()

    # сокращение дроби
    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

    # прочитать значение дроби с клавиатуры. дробь вводится как a/b
    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()

    # Вывести дробь на экран

    def display(self):
        print(f"{self.__numerator}/{self.__denominator}")
    # Сложение обыкновенных дробей.
```

```

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()
    # Вычитание обыкновенных дробей.

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()

    # Умножение обыкновенных дробей.
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()

    # Деление обыкновенных дробей.
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()

    # Отношение обыкновенных дробей.
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()
```

### Результат выполнения программы:

```
3/4
Введите обыкновенную дробь: 6/8
3/4
3/2
0/1
9/16
1/1
```

Рисунок 1. Результат работы программы

### Выполнение задания.

### Код программы:

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

class Para:
    def __init__(self, first, second):
        if isinstance(first, float) and isinstance(second, float):
            self.first = first
            self.second = second
        else:
            print("Ошибка! Значения полей должны быть дробными числами.")

    def read(self):
        self.first = float(input("Введите значение первого поля: "))
        self.second = float(input("Введите значение второго поля: "))

    def display(self):
        print("Значение первого поля:", self.first)
        print("Значение второго поля:", self.second)

    def root(self):
        if self.second != 0:
            return -self.first / self.second
        else:
            print("Ошибка! Коэффициент В не может быть равен нулю.")

def make_para(first, second):
    return Para(first, second)
```

```
if __name__ == "__main__":  
    para = make_para(2.5, 3.5)  
    para.display()  
  
    rootv = para.root()  
    if rootv is not None:  
        print("Корень линейного уравнения:", rootv)  
  
    para.read()  
    para.display()  
  
    rootv = para.root()  
    if rootv is not None:  
        print("Корень линейного уравнения:", rootv)
```

### Результат выполнения программы:

```
Значение первого поля: 2.5  
Значение второго поля: 3.5  
Корень линейного уравнения: -0.7142857142857143  
Введите значение первого поля: 2.8  
Введите значение второго поля: 2.6  
Значение первого поля: 2.8  
Значение второго поля: 2.6  
Корень линейного уравнения: -1.0769230769230769
```

Рисунок 2. Результат выполнения

## Задание повышенной сложности.

### Код программы:

```
class Bancomat:
    def __init__(self, number, maxs, mins):
        self.banknotes = {"10": 0, "100": 0, "500": 0, "1000": 0}
        self.number = number
        self.maxs = maxs
        self.mins = mins

    @property
    def total_sum(self):
        total = 0
        for nominal, count in self.banknotes.items():
            total += (int(nominal) * count)
        return total

    def LoadMoney(self, tuple):
        for nominal, count in tuple.items():
            self.banknotes[nominal] += count
        print("Остаток в банкомате: ", self.total_sum)

    def GetMoney(self, sum_tuple):
        if self.total_sum >= self.CheckSum(sum_tuple) >= self.mins and self.CheckSum(sum_tuple) <= self.maxs:
            for nominal, count in sum_tuple.items():
                self.banknotes[nominal] -= count
            print("Остаток в банкомате:", self.total_sum)
        else:
            print("Error!")

    def CheckSum(self, tuple):
        total = 0
        for nominal, count in tuple.items():
            total += (int(nominal) * count)
        return total

if __name__ == "__main__":
    obj = Bancomat("123", 1000, 100)
    summ = {"10": 1, "1000": 1, "500": 20}
    obj.LoadMoney(summ)
    print(obj.banknotes)
    summ = {"10": 1, "1000": 0, "500": 1}
    obj.GetMoney(summ)
    print(obj.banknotes)
```

### Результат выполнения программы:

```
Остаток в банкомате: 11010
{'10': 1, '100': 0, '500': 20, '1000': 1}
Остаток в банкомате: 10500
{'10': 0, '100': 0, '500': 19, '1000': 1}
```

Рисунок 3. Результат работы программы

### Ответы на контрольные вопросы:

### Контрольные вопросы:

## **1. Как осуществляется объявление класса в языке Python?**

Для создания класса в Python используется инструкция `class`. Она сильно похожа на объявление функций `def` и так же, как и `def`, `class` создаёт объект.

Инструкция `class` имеет следующий синтаксис:

```
**class <Name> ([<Superclass1>], [<Superclass2>]):
```

```
<name declarations>**
```

## **2. Чем атрибуты класса отличаются от атрибутов экземпляра?**

Атрибуты экземпляра и класса отличаются способом получения доступа к ним. Другими словами, речь идет об использовании названия класса и использовании названия экземпляра. С другой стороны, глобальные и локальные переменные отличаются своими областями видимости, другими словами, местами, где к ним может быть получен доступ.

## **3. Каково назначение методов класса?**

Методы определяют набор действий, которые доступны классу (часто говорят, что они определяют поведение класса). Метод описывается один раз, а может вызываться для различных объектов класса столько раз, сколько необходимо. Общий формат записи методов класса имеет следующий вид:

```
[атрибуты] [спецификаторы] тип метода имя метода  
([параметры]) {тело метода}.
```

## **4. Для чего предназначен метод `__init__()` класса?**

Метод `__init__` в определении класса позволяет нам инициализировать атрибуты или переменные экземпляра для всех экземпляров класса. Метод `__init__` вызывается каждый раз, когда создается новый экземпляр класса. Цель наличия нескольких методов `__init__` в классе Python – предоставить несколько конструкторов для создания объектов.

## **5. Каково назначение `self` ?**

Ключевое слово `self` в Python используется для ссылки на

текущий экземпляр объекта класса. Оно обычно используется в методах класса, чтобы обращаться к атрибутам и методам этого объекта. Когда мы вызываем метод объекта класса, Python автоматически передает ссылку на этот объект в качестве первого аргумента метода, который мы обычно называем `self`. Таким образом, мы можем обращаться к атрибутам и методам объекта через `self`, как в примере выше, где мы сохраняем имя объекта в атрибуте `name` и выводим его через метод `say_hello`.

## **6. Как добавить атрибуты в класс?**

Атрибуты могут быть добавлены в класс путем определения их внутри класса.

Например:

```
class MyClass:

    def __init__(self, attribute1, attribute2):

        self.attribute1 = attribute1

        self.attribute2 = attribute2
```

## **7. Как осуществляется управление доступом к методам и атрибутам в языке Python?**

Управление доступом к методам и атрибутам в языке Python осуществляется с помощью модификаторов доступа. В Python есть три уровня доступа: `public`, `protected` и `private`.

## **8. Каково назначение функции `isinstance`?**

Функция `isinstance` в языке Python используется для проверки принадлежности объекта определенному классу. Она принимает два аргумента: объект и класс, и возвращает `True`, если объект принадлежит к указанному классу или его наследникам, и `False` в противном случае. Функция `isinstance` может быть полезна, например, при проверке типов аргументов функции или при обработке объектов разных классов в цикле

**Вывод:** в ходе работы были приобретены навыки работы с классами и объектами с помощью языка программирования Python версии 3.x.