# МИНИCTEPCTBO 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

Элементы объектно-ориентированного программирования в языке Python.

|  |
| --- |
| Выполнил студент группы  ИВТ-б-о-21-1 |
| Богдашов Артём . «» 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:

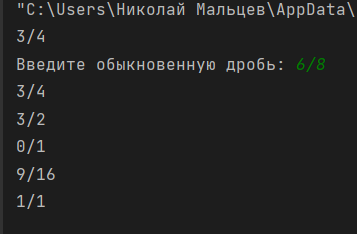


Рисунок 1. Результат работы программы

# Выполнение задания. Код программы:

class Pair:  
 def \_\_init\_\_(self, first, second):  
 if not isinstance(first, int) or not isinstance(second, int):  
 raise ValueError("первый и второй должны быть целыми числами ")  
 if second == 0:  
 raise ValueError("'Секунда не равна нулю")  
  
 self.first = first  
 self.second = second  
  
 def ipart(self):  
 return self.first // self.second  
  
 def display(self):  
 print(f"{self.first}/{self.second}")  
  
 @classmethod  
 def read(cls):  
 try:  
 first = int(input("Введите сначало (числитель): "))  
 second = int(input("Введите (знаменатель):: "))  
 return cls(first, second)  
 except ValueError:  
 print("Ошибка ввода. Пожалуйста введите целочисленное число 'первое' и 'второе'")  
 return None  
  
  
def make\_pair():  
 while True:  
 try:  
 pair = Pair.read()  
 if pair is not None:  
 return pair  
 except KeyboardInterrupt:  
 print("Operation aborted by user.")  
 return None  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 fraction = make\_pair()  
 if fraction:  
 fraction.display()  
 print("Integer part:", fraction.ipart())

**Результат выполнения программы:**

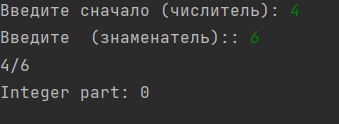


Рисунок 2. Результат выполнения

# Второе задание.

# Код программы:

class Money:  
 def \_\_init\_\_(self, rubles, kopecks):  
 self.rubles = rubles  
 self.kopecks = kopecks  
  
 def read(self):  
 try:  
 self.rubles = int(input("Введите количество рублей: "))  
 self.kopecks = int(input("Введите количество копеек: "))  
 except ValueError:  
 print("Некорректный ввод. Введите целое число.")  
  
 def display(self):  
 print(f"{self.rubles} руб. {self.kopecks} коп.")  
  
 def \_\_add\_\_(self, other):  
 total\_kopecks = self.rubles \* 100 + self.kopecks + other.rubles \* 100 + other.kopecks  
 new\_rubles = total\_kopecks // 100  
 new\_kopecks = total\_kopecks % 100  
 return Money(new\_rubles, new\_kopecks)  
  
 def \_\_sub\_\_(self, other):  
 total\_kopecks = self.rubles \* 100 + self.kopecks - (other.rubles \* 100 + other.kopecks)  
 new\_rubles = total\_kopecks // 100  
 new\_kopecks = total\_kopecks % 100  
 return Money(new\_rubles, new\_kopecks)  
  
 def \_\_truediv\_\_(self, other):  
 if other == 0:  
 raise ValueError("Деление на ноль недопустимо.")  
 total\_kopecks = self.rubles \* 100 + self.kopecks  
 result = total\_kopecks / other  
 new\_rubles = int(result)  
 new\_kopecks = round((result - new\_rubles) \* 100)  
 return Money(new\_rubles, new\_kopecks)  
  
 def \_\_mul\_\_(self, other):  
 total\_kopecks = self.rubles \* 100 + self.kopecks  
 result = total\_kopecks \* other  
 new\_rubles = int(result // 100)  
 new\_kopecks = int(result % 100)  
 return Money(new\_rubles, new\_kopecks)  
  
 def \_\_eq\_\_(self, other):  
 return (self.rubles, self.kopecks) == (other.rubles, other.kopecks)  
  
 def \_\_lt\_\_(self, other):  
 return (self.rubles, self.kopecks) < (other.rubles, other.kopecks)  
  
 def \_\_le\_\_(self, other):  
 return (self.rubles, self.kopecks) <= (other.rubles, other.kopecks)  
  
 def \_\_str\_\_(self):  
 return f"{self.rubles} руб. {self.kopecks} коп."  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 money1 = Money(10, 50)  
 money2 = Money(5, 75)  
  
 print("Деньги 1:")  
 money1.display()  
  
 print("Деньги 2:")  
 money2.display()  
  
 # Примеры операций  
 print("\nОперации:")  
 result = money1 + money2  
 print("Сумма:")  
 result.display()  
  
 result = money1 - money2  
 print("Разность:")  
 result.display()  
  
 result = money1 / 2.5  
 print("Деление на число:")  
 result.display()  
  
 result = money1 \* 3  
 print("Умножение на число:")  
 result.display()  
  
 print("Сравнение:")  
 print(f"money1 равно money2: {money1 == money2}")  
 print(f"money1 меньше money2: {money1 < money2}")

**Результат выполнения программы:**

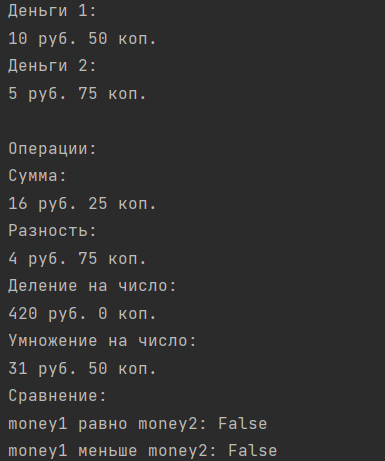


Рисунок 3. Результат работы программы

# 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>\*\*

# 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.

# 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}.

# 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.

# 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.

# 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

# 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.

# 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.