# 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 №2.17

Элементы объектно-ориентированного программирования в языке Python.

|  |
| --- |
| Выполнил студент группы  ИВТ-б-о-21-1 |
| Урусов М.А. « » 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:

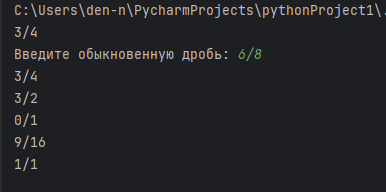
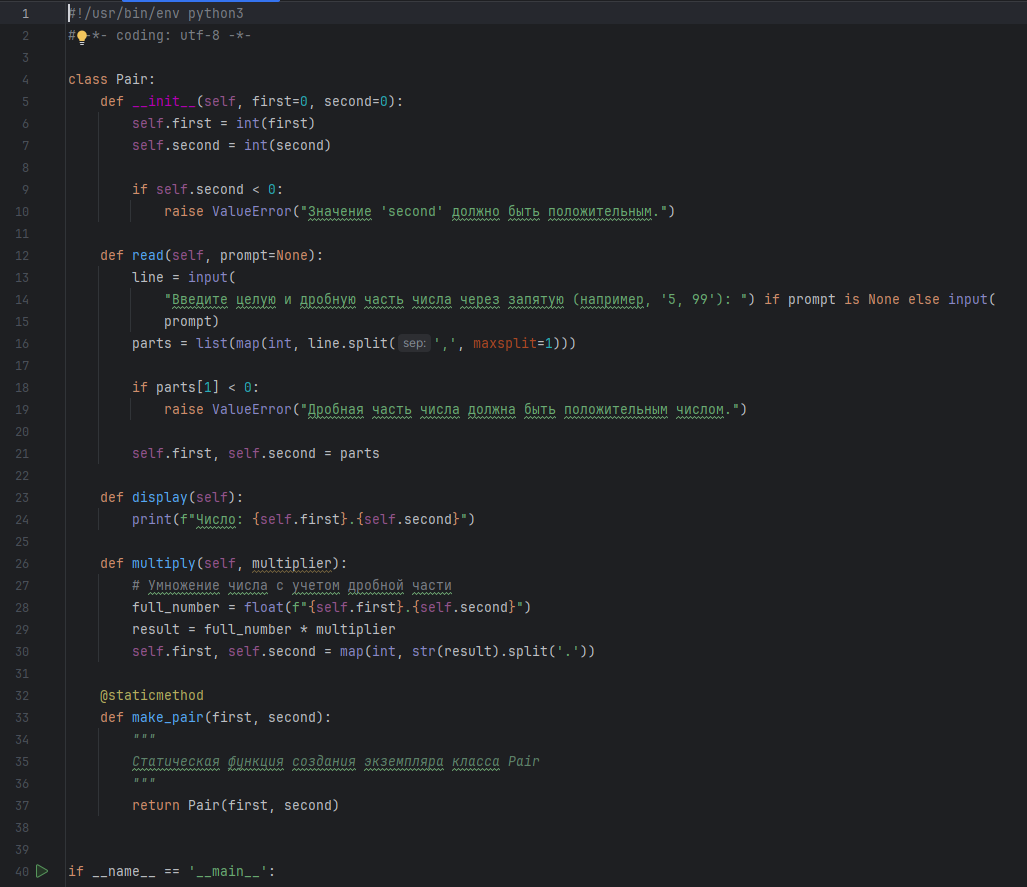
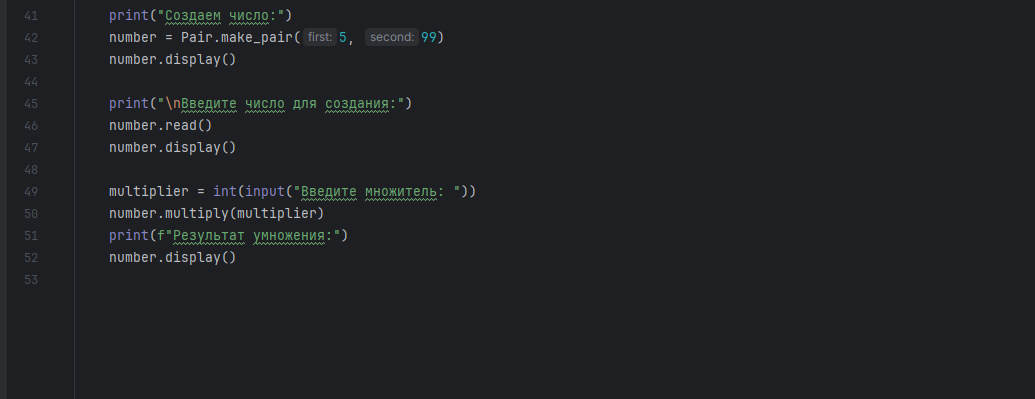


Рисунок 1. Результат работы программы

# Выполнение задания. Код программы:

****

**Результат выполнения программы:**

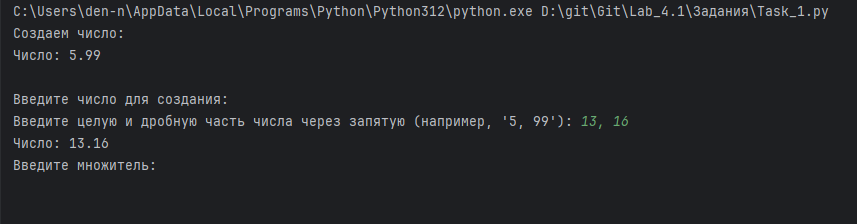


Рисунок 2. Результат выполнения

# Задание повышенной сложности. Код программы:

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
  
import argparse  
import json  
import os.path  
import pathlib  
import logging  
from datetime import datetime  
  
# Настройка логгирования с добавлением времени выполнения до миллисекунд  
logging.basicConfig(filename='planes.log', level=logging.INFO,  
 format='%(asctime)s.%(msecs)03d - %(levelname)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')  
  
  
def add\_plane(staff, destination, num, typ):  
 staff.append({"destination": destination, "num": num, "typ": typ})  
 logging.info(f"Добавлен самолет: пункт назначения {destination}, номер {num}, тип {typ}")  
 return staff  
  
  
def display\_planes(staff):  
 if staff:  
 line = '+-{}-+-{}-+-{}-+-{}-+'.format('-' \* 4, '-' \* 30, '-' \* 20, '-' \* 15)  
 print(line)  
 print('| {:^4} | {:^30} | {:^20} | {:^15} |'.format("No", "Пункт назначения", "Номер рейса", "Тип самолета"))  
 print(line)  
 for idx, plane in enumerate(staff, 1):  
 print('| {:>4} | {:<30} | {:<20} | {:>15} |'.format(idx, plane.get('destination', ''), plane.get('num', ''),  
 plane.get('typ', '')))  
 print(line)  
 else:  
 print("Список самолетов пуст")  
 logging.info("Попытка отобразить пустой список самолетов")  
  
  
def select\_planes(staff, jet):

result = [plane for plane in staff if jet == plane.get('typ', '')]  
 if not result:  
 logging.info(f"Самолеты типа {jet} не найдены")  
 return result  
  
  
def save\_planes(file\_name, staff):  
 try:  
 with open(file\_name, "w", encoding="utf-8") as fout:  
 json.dump(staff, fout, ensure\_ascii=False, indent=4)  
 logging.info(f"Данные успешно сохранены в файл {file\_name}")  
 except Exception as e:  
 logging.error(f"Ошибка при сохранении данных: {e}")  
  
  
def load\_planes(file\_name):  
 try:  
 with open(file\_name, "r", encoding="utf-8") as fin:  
 return json.load(fin)  
 except FileNotFoundError:  
 logging.warning(f"Файл {file\_name} не найден. Будет создан новый файл.")  
 return []  
 except Exception as e:  
 logging.error(f"Ошибка при загрузке данных: {e}")  
 return []  
  
  
def main(command\_line=None):  
 start\_time = datetime.now()  
  
 parser = argparse.ArgumentParser("planes")  
 subparsers = parser.add\_subparsers(dest="command")  
  
 file\_parser = argparse.ArgumentParser(add\_help=False)  
 file\_parser.add\_argument("filename", action="store", help="The data file name")  
  
 add = subparsers.add\_parser("add", parents=[file\_parser], help="Add a new plane")  
 add.add\_argument("-d", "--destination", action="store", required=True, help="The plane's destination")  
 add.add\_argument("-n", "--num", action="store", type=int, required=True, help="The plane's number")  
 add.add\_argument("-t", "--typ", action="store", required=True, help="The plane's type")  
  
 \_ = subparsers.add\_parser("display", parents=[file\_parser], help="Display all planes")  
  
 select = subparsers.add\_parser("select", parents=[file\_parser], help="Select the planes")  
 select.add\_argument("-T", "--type", action="store", required=True, help="The required type")  
  
 args = parser.parse\_args(command\_line)  
 is\_dirty = False  
 args.filename = pathlib.Path.home().joinpath(args.filename)  
 if os.path.exists(args.filename):  
 planes = load\_planes(args.filename)  
 else:  
 planes = []  
  
 if args.command == "add":  
 planes = add\_plane(planes, args.destination, args.num, args.typ)  
 is\_dirty = True  
 elif args.command == "display":  
 display\_planes(planes)  
 elif args.command == "select":  
 selected = select\_planes(planes, args.type)  
 display\_planes(selected)  
  
 if is\_dirty:  
 save\_planes(args.filename, planes)  
  
 end\_time = datetime.now()  
 logging.info(f"Выполнение команды завершено. Время выполнения: {end\_time - start\_time}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Результат выполнения программы:**

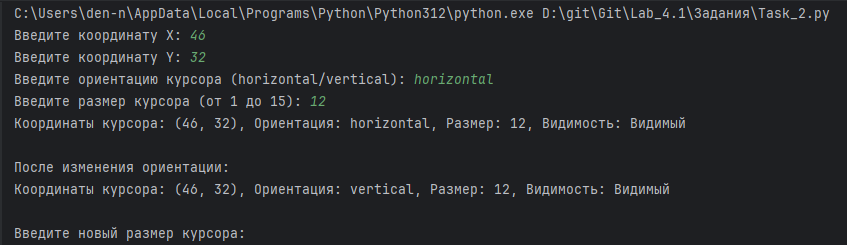


Рисунок 3. Результат работы программы

# 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