

**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 №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

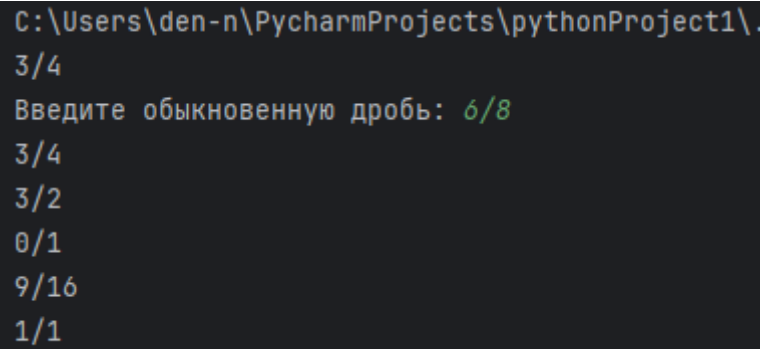
def less(self, rhs):
    if isinstance(rhs, Rational):
        v1 = self.numerator / self.denominator
        v2 = rhs.numerator / rhs.denominator
        return v1 < v2
    else:
        return False

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:



```
C:\Users\den-n\PycharmProjects\pythonProject1\
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:

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  class Pair:
5      def __init__(self, first=0, second=0):
6          self.first = int(first)
7          self.second = int(second)
8
9          if self.second < 0:
10             raise ValueError("Значение 'second' должно быть положительным.")
11
12     def read(self, prompt=None):
13         line = input(
14             "Введите целую и дробную часть числа через запятую (например, '5, 99'): ") if prompt is None else input(
15             prompt)
16         parts = list(map(int, line.split(sep=',', maxsplit=1)))
17
18         if parts[1] < 0:
19             raise ValueError("Дробная часть числа должна быть положительным числом.")
20
21         self.first, self.second = parts
22
23     def display(self):
24         print(f"Число: {self.first}.{self.second}")
25
26     def multiply(self, multiplier):
27         # Умножение числа с учетом дробной части
28         full_number = float(f"{self.first}.{self.second}")
29         result = full_number * multiplier
30         self.first, self.second = map(int, str(result).split('.'))
31
32     @staticmethod
33     def make_pair(first, second):
34         """
35         Статическая функция создания экземпляра класса Pair
36         """
37         return Pair(first, second)
38
39
40 if __name__ == '__main__':

```

```

41     print("Создаем число:")
42     number = Pair.make_pair( first=5, second=99)
43     number.display()
44
45     print("\nВведите число для создания:")
46     number.read()
47     number.display()
48
49     multiplier = int(input("Введите множитель: "))
50     number.multiply(multiplier)
51     print(f"Результат умножения:")
52     number.display()
53

```

Результат выполнения программы:

```
C:\Users\den-n\AppData\Local\Programs\Python\Python312\python.exe D:\git\6it\Lab_4.1\Задания\Task_1.py
Создаем число:
Число: 5.99

Введите число для создания:
Введите целую и дробную часть числа через запятую (например, '5, 99'): 13, 16
Число: 13.16
Введите множитель:
```

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:
```

Результат выполнения программы:

```
C:\Users\den-n\AppData\Local\Programs\Python\Python312\python.exe D:\git\Git\Lab_4.1\Задания\Task_2.py
Введите координату X: 46
Введите координату Y: 32
Введите ориентацию курсора (horizontal/vertical): horizontal
Введите размер курсора (от 1 до 15): 12
Координаты курсора: (46, 32), Ориентация: horizontal, Размер: 12, Видимость: Видимый

После изменения ориентации:
Координаты курсора: (46, 32), Ориентация: vertical, Размер: 12, Видимость: Видимый

Введите новый размер курсора:
```

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`.

1. 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
```

2. 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`.

3. 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

