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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ

По лабораторной работе №1

Дисциплины «Объектно-ориентированное программирование»

Выполнил:

Пустяков Андрей Сергеевич

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной
техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А., доцент департамента цифровых и робототехнических систем и электроники института перспективной инженерии

_____

(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Элементы объектно-ориентированного программирования на языке Python.

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

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git flow» (рис. 1).

```
C:\Program Files\Git>cd C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_1

C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_1>git clone https://github.com/AndreyPust/Object-Oriented_Prograr
Cloning into 'Object-Oriented_Programming_laboratory_work_1'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), done.

C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_1>cd C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_1\Object-Or
_work_1

C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_1\Object-Oriented_Programming_laboratory_work_1>git branch
* main

C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_1\Object-Oriented_Programming_laboratory_work_1>git flow init

Which branch should be used for bringing forth production releases?
   - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]
```

```
C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_1\Object-Oriented_Programming_laboratory_work_1>git flow feature start feature_branch/0.1
Switched to a new branch 'feature/feature_branch/0.1'

Summary of actions:
- A new branch 'feature/feature_branch/0.1' was created, based on 'develop'
- You are now on branch 'feature/feature_branch/0.1'

Now, start committing on your feature. When done, use:

     git flow feature finish feature_branch/0.1


C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_1\Object-Oriented_Programming_laboratory_work_1>git branch
  develop
* feature/feature_branch/0.1
  main
```

Рисунок 1 – Создание репозитория

Проработка примеров лабораторной работы:

Пример 1.

Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где a – числитель, b – знаменатель. Необходимо создать класс Rational

для работы с рациональными дробями. Обязательно должны быть реализованы операции:

- сложения add, (a, b);
- вычитания sub, c ;
- умножения mul, ;
- деления div, ;
- сравнения equal, greate, less.

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

Код программы, для решения данного задания (рис. 2).

```
#!/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()
```

Рисунок 2 – Код программы с реализацией класса для работы с дробями

Результаты работы программы (для примера берем дробь 5/6) (рис. 3).

Рисунок 3 – Результаты работы программы

Выполнение индивидуальных заданий:

Вариант 25

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.

Поле «first» – дробное положительное число, цена товара; поле «second» – целое положительное число, количество единиц товара. Реализовать метод «cost()» — вычисление стоимости товара (Вариант 25(5)).

Код программы решения индивидуального задания 1:

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

# Поле «first» – дробное положительное число, цена товара;
# поле «second» – целое положительное число, количество единиц товара.
# Реализовать метод «cost()» — вычисление стоимости товара (Вариант 25(5)).

class Pair:
    def __init__(self, first, second):
        """
        Метод инициализации, проверяет аргументы на корректность.
        """

        if not isinstance(first, (int, float)) or first <= 0:
            raise ValueError("Цена товара должна быть положительным дробным
числом!")
        if not isinstance(second, int) or second <= 0:
            raise ValueError("Количество товара должно быть положительным
целым числом!")

        self.first = float(first)  # Цена товара
        self.second = int(second)  # Количество товара

    def get_first(self):
        return self.first

    def get_second(self):
        return self.second


class PairInput:
    @staticmethod
    def read():
        """
        Метод, позволяющий производить ввод значений с клавиатуры.
        """

        try:
            first = float(input("Введите цену товара (положительное дробное
число): "))
            if first <= 0:
                raise ValueError("Цена товара должна быть положительным
числом!")

            second = int(input("Введите количество товара (положительное
целое число): "))
            if second <= 0:
                raise ValueError("Количество товара должно быть положительным
целым числом!")

            return Pair(first, second)  # Возвращаем объект Pair с введенными
данными
        except ValueError as e:
            print(f"Ошибка ввода: {e}")
            return None


class PairDisplay:
    @staticmethod
    def display(any_pair):
        """
        Метод, позволяющий выводить информацию о товаре (количество,
```

```
стоимость).
        """

        if isinstance(any_pair, Pair):
            print(f"Цена товара: {any_pair.get_first()}, Количество:
{any_pair.get_second()}")
        else:
            print("Невозможно вывести данные, объект пары некорректен.")


class PairCostCalculator:
    @staticmethod
    def cost(any_pair):
        """
        Метод, вычисляющий стоимость всего товара.
        """

        if isinstance(any_pair, Pair):
            return any_pair.get_first() * any_pair.get_second()
        else:
            raise ValueError("Некорректный объект пары для вычисления
стоимости!")


def make_pair(first, second):
    """
    Функция для создания объекта Pair с проверкой на корректность данных
    """

    try:
        return Pair(first, second)
    except ValueError as e:
        print(f"Ошибка создания объекта: {e}")
        return None


# Пример использования
if __name__ == "__main__":
    # Создание объекта через функцию make_pair
    pair = make_pair(9.99, 5)

    if pair:
        PairDisplay.display(pair)  # Вывод данных
        print(f"Общая стоимость : {PairCostCalculator.cost(pair)}")  # Расчет
стоимости всего товара

    # Способ с вводом данных с клавиатуры
    new_pair = PairInput.read()  # Ввод данных и создание новой пары
    if new_pair:
        PairDisplay.display(new_pair)  # Вывод данных
        print(f"Общая стоимость : {PairCostCalculator.cost(new_pair)}")  #
Расчет стоимости всего товара
```

Результаты работы программы с введенными в программе значениями стоимости и количества и с использованием метода «read()» (рис. 4).

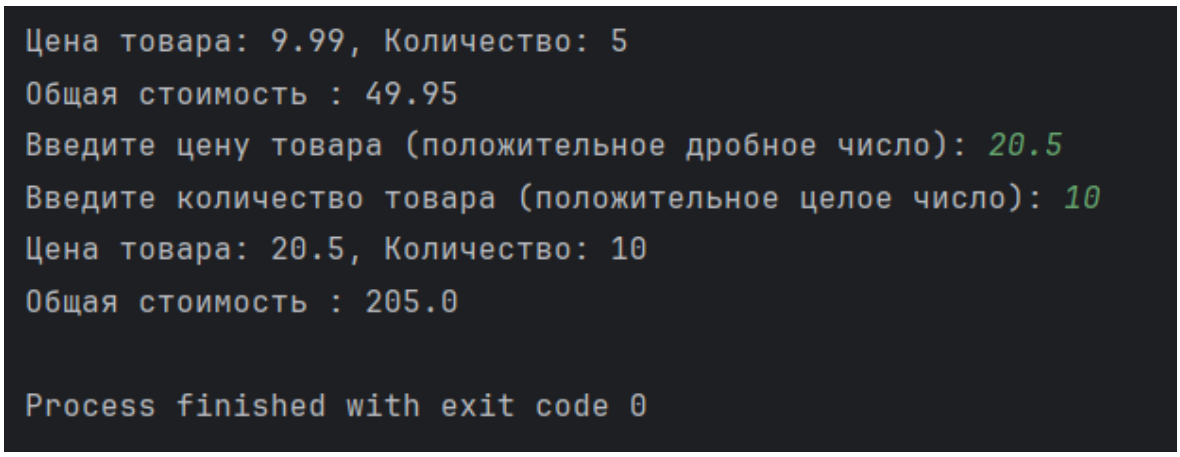

Рисунок 4 – Результаты работы программы

Результаты работы программы в случае, если данные были введены некорректно (рис. 5).

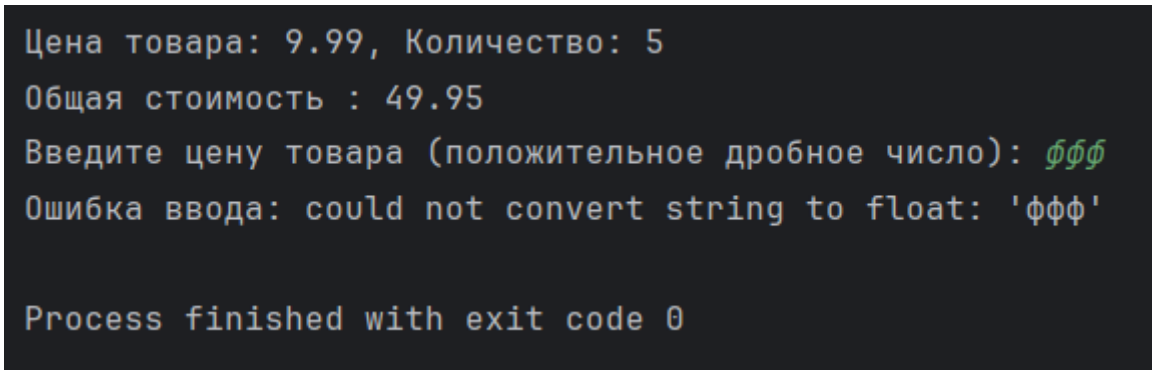

Рисунок 5 – Результаты работы программы при неверном вводе

Задание 2.

Необходимо 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».

В раздел программы, начинающийся после инструкции «if __name__ = '__main__':» добавить код, демонстрирующий возможности разработанного класса.

Номиналы российских рублей могут принимать значения 1, 2, 5, 10, 50, 100, 500, 1000, 5000. Копейки представить как 0.01 (1 копейка), 0.05 (5 копеек),

0.1 (10 копеек), 0.5 (50 копеек). Создать класс Money для работы с денежными суммами. Сумма должна быть представлена полями-номиналами, значениями которых должно быть количество купюр данного достоинства. Реализовать сложение сумм, вычитание сумм, деление сумм, деление суммы на дробное число, умножение на дробное число и операции сравнения. Дробная часть (копейки) при выводе на экран должны быть отделена от целой части запятой (Вариант 25 (10)).

Код программы решения индивидуального задания 2:

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

# Номиналы российских рублей могут принимать значения 1, 2, 5, 10, 50, 100,
500, 1000, 5000.
# Копейки представить как 0.01 (1 копейка), 0.05 (5 копеек), 0.1 (10 копеек),
0.5 (50 копеек).
# Создать класс Money для работы с денежными суммами. Сумма должна быть
представлена полями-номиналами,
# значениями которых должно быть количество купюр данного достоинства.
Реализовать сложение сумм,
# вычитание сумм, деление сумм, деление суммы на дробное число, умножение на
дробное число и операции сравнения.
# Дробная часть (копейки) при выводе на экран должны быть отделена от целой
части запятой (Вариант 25 (10)).

class Money:
    # Список номиналов рублевых купюр и копеек
    denominations = {
        5000: '5000 руб.',
        1000: '1000 руб.',
        500: '500 руб.',
        100: '100 руб.',
        50: '50 руб.',
        10: '10 руб.',
        5: '5 руб.',
        2: '2 руб.',
        1: '1 руб.',
        0.5: '50 коп.',
        0.1: '10 коп.',
        0.05: '5 коп.',
        0.01: '1 коп.'
    }

    def __init__(self):
        # Инициализация поля для каждого номинала
        self.amounts = {denom: 0 for denom in self.denominations}

    def read(self):
        # Ввод количества купюр и монет с клавиатуры
        try:
            for denom in self.denominations:
                self.amounts[denom] = int(input(f"Введите количество
{self.denominations[denom]}: "))
        except ValueError:
```

```
            print("Ошибка ввода. Все значения должны быть целыми числами.")

    def display(self):
        # Выводим информацию о сумме
        total_rubles, total_kopeks = self.total_value()
        print(f"Сумма: {total_rubles},{int(total_kopeks):02d} руб.")

    def total_value(self):
        # Рассчитываем общую сумму денег в рублях и копейках
        total = 0
        for denom, count in self.amounts.items():
            total += denom * count

        rubles = int(total)
        kopeks = round((total - rubles) * 100)
        return rubles, kopeks

    def __add__(self, other):
        # Сложение двух денежных сумм
        result = Money()
        for denom in self.denominations:
            result.amounts[denom] = self.amounts[denom] +
other.amounts[denom]
        return result

    def __sub__(self, other):
        # Вычитание двух денежных сумм
        result = Money()
        for denom in self.denominations:
            result.amounts[denom] = self.amounts[denom] -
other.amounts[denom]
        return result

    def __truediv__(self, number):
        # Деление суммы на дробное число
        result = Money()
        for denom in self.denominations:
            result.amounts[denom] = int(self.amounts[denom] / number)
        return result

    def __mul__(self, number):
        # Умножение суммы на дробное число
        result = Money()
        for denom in self.denominations:
            result.amounts[denom] = int(self.amounts[denom] * number)
        return result

    def __eq__(self, other):
        # Операция сравнения "равно"
        return self.total_value() == other.total_value()

    def __lt__(self, other):
        # Операция сравнения "меньше"
        return self.total_value() < other.total_value()

    def __le__(self, other):
        # Операция сравнения "меньше или равно"
        return self.total_value() <= other.total_value()


if __name__ == "__main__":
    # Демонстрация работы класса Money

    # Ввод первой суммы с клавиатуры
```

```
print("Введите первую сумму:")
money1 = Money()
money1.read()

# Ввод второй суммы с клавиатуры
print("\nВведите вторую сумму:")
money2 = Money()
money2.read()

# Вывод суммы на экран
print("\nПервая сумма:")
money1.display()

print("Вторая сумма:")
money2.display()

# Сложение сумм
sum_result = money1 + money2
print("\nРезультат сложения:")
sum_result.display()

# Вычитание сумм
sub_result = money1 - money2
print("\nРезультат вычитания:")
sub_result.display()

# Деление суммы на число
div_result = money1 / 2
print("\nРезультат деления первой суммы на 2:")
div_result.display()

# Умножение суммы на число
mul_result = money1 * 1.5
print("\nРезультат умножения первой суммы на 1.5:")
mul_result.display()

# Сравнение сумм
if money1 == money2:
    print("\nСуммы равны.")
elif money1 < money2:
    print("\nПервая сумма меньше второй.")
else:
    print("\nПервая сумма больше второй.")
```

Ответы на контрольные вопросы:

1. Как осуществляется объявление класса в языке Python?

В Python объявление класса осуществляется с помощью ключевого слова «class», после которого следует название класса (как правило согласно стандарту «Camel Case» названия классов начинаются с заглавной буквы, как и все слова во фразе).

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибут класса – это атрибут (данные внутри классов), общий для всех экземпляров класса, к ним можно получить доступ использовав точечную запись с именем класса. Атрибут класса определен внутри класса, вне каких-либо методов (значения одинаковы для всех экземпляров этого класса).

Атрибут экземпляра же хранит информацию, уникальную для каждого экземпляра класса. Как правило атрибуты экземпляра создаются в методе «__init__()», поскольку он является конструктором. Атрибуты экземпляра доступны только из области видимости объекта.

3. Каково назначение методов класса?

Методы определяют функциональность объектов, принадлежащих конкретному классу. Методы класса в Python выполняют различные функции, в зависимости от их типа и назначения. Они позволяют объектам класса взаимодействовать с данными и выполнять определенные действия. К примеру, осуществлять взаимодействие с данными конкретного объекта или выполнять логику, связанную с состоянием объекта.

4. Для чего предназначен метод __init__() класса?

Метод «__init__()» - это конструктор классов, который предназначен для создания экземпляров класса. Если этот метод определен внутри класса, то он автоматически вызывается при создании нового экземпляра класса. В нем задаются, какие атрибуты будут у будущих экземпляров класса.

5. Каково назначение self?

Аргумент «self» предназначен для получения доступа к атрибутам и методам класса (представляет конкретный экземпляр класса). Параметр «self» передается методу экземпляра неявно при его вызове.

6. Как добавить атрибуты в класс?

Для создания атрибута в класс его необходимо объявить внутри класса и вне какого-либо метода. Для добавления атрибута в уже существующий класс можно использовать методы класса (например, с помощью метода «setattr»). Также можно добавить атрибуты в класс с помощью наследования (от родительского в дочерний).

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

Управление доступом к методам и атрибутам в Python осуществляется в основном через соглашения об именовании (одно подчеркивание — защита, два подчеркивания — приватность) и с помощью механизмов, таких как свойства (properties), позволяющих контролировать доступ и изменения данных через методы.

8. Каково назначение функции isinstance?

Функция «isinstance()» в Python используется для проверки, является ли объект экземпляром указанного класса или его подклассов. Она возвращает «True», если объект принадлежит указанному классу (или его подклассу), и «False» в противном случае.

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