

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

Департамент цифровых, робототехнических систем и электроники института
перспективной инженерии

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4.1
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Юрьев Илья Евгеньевич
курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Хацукова А.И., ассистент
департамента цифровых
робототехнических систем и
электроники

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: элементы объектно-ориентированного программирования в языке

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

Ход выполнения работы:

создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и .gitignore файл для языка программирования Python:

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * **Repository name ***

daxstrong / OOP_1

✔ OOP_1 is available.

Great repository names are short and memorable. Need inspiration? How about [super-duper-octo-winner](#) ?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

ⓘ You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

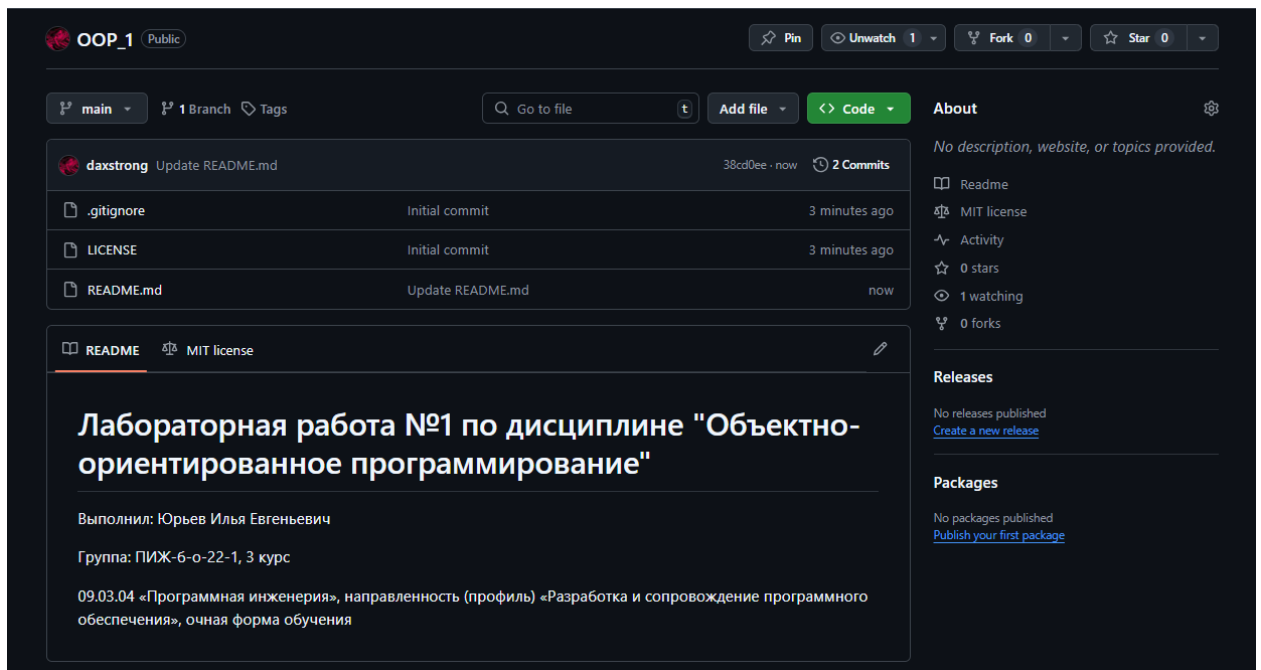


Рисунок 2 – Созданный репозиторий

```
C:\Users\Ilya\Documents\labs\OOP>git clone https://github.com/daxstrong/OOP_1.git
Cloning into 'OOP_1'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (8/8), 4.16 KiB | 1.04 MiB/s, done.
Resolving deltas: 100% (1/1), done.

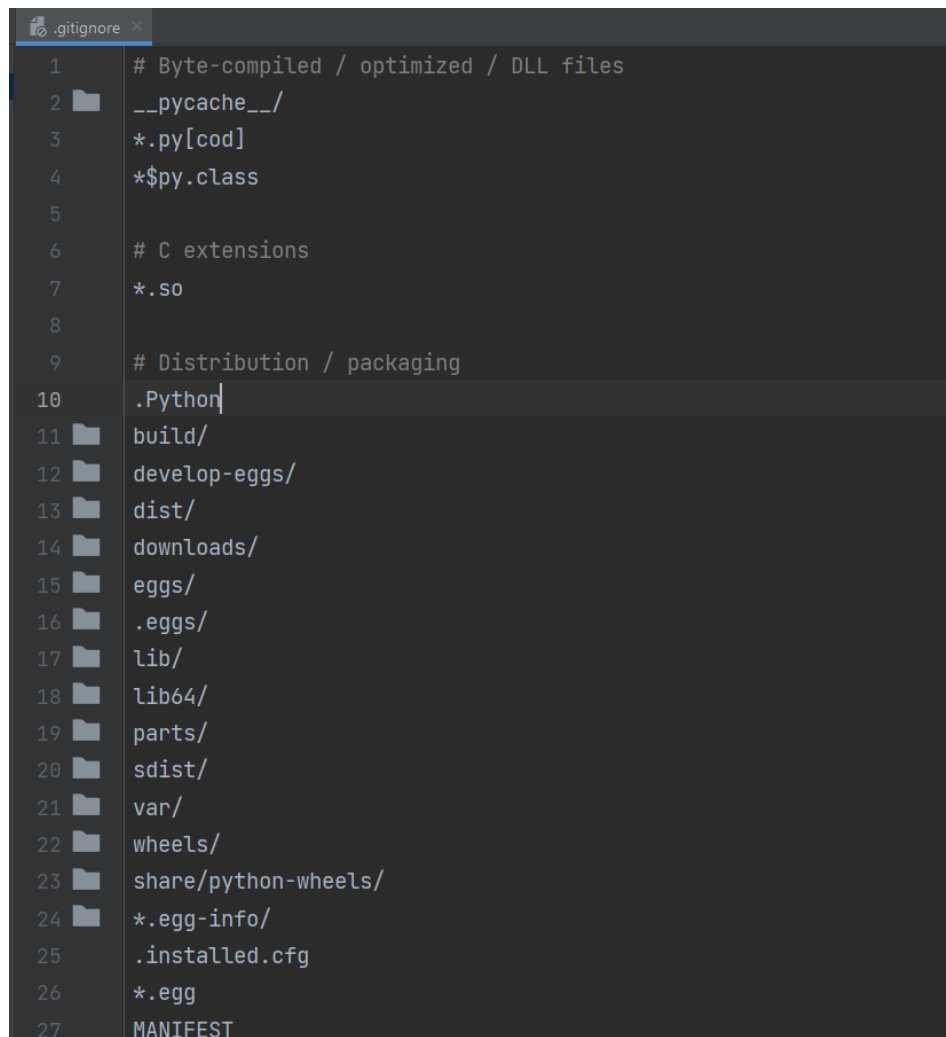
C:\Users\Ilya\Documents\labs\OOP>|
```

Рисунок 3 – Клонирование репозитория

```
C:\Users\Ilya\Documents\labs\OOP\OOP_1>git checkout -b develop
Switched to a new branch 'develop'

C:\Users\Ilya\Documents\labs\OOP\OOP_1>|
```

Рисунок 4 – Создание ветки разработки, в которой будут вноситься изменения до окончательного релиза проекта



The image shows a code editor window with a file named `.gitignore`. The file contains a list of patterns to ignore, organized into sections with comments. The patterns include byte-compiled files, C extensions, distribution/packaging files, and various build and development directories.

```
1 # Byte-compiled / optimized / DLL files
2 __pycache__/
3 *.py[cod]
4 *$py.class
5
6 # C extensions
7 *.so
8
9 # Distribution / packaging
10 .Python|
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
```

Рисунок 5 – Часть `.gitignore`, созданного GitHub

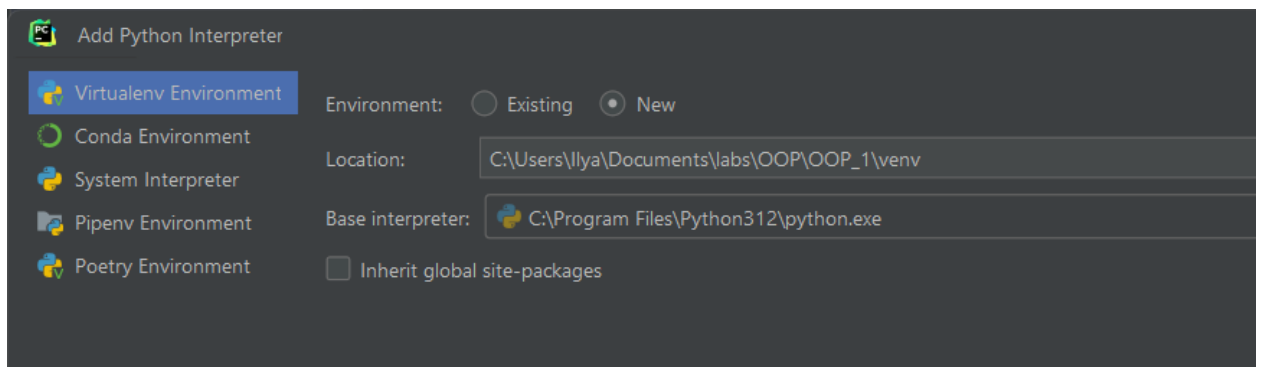


Рисунок 6 – Создание виртуального окружения

Try the redesigned packaging support in Python Packages tool window. [Go to tool window](#) ×

Package	Version	Latest version
pip	23.2.1	↑ 24.3.1
setuptools	68.2.0	↑ 75.3.0
wheel	0.41.2	↑ 0.44.0

Рисунок 7 – Созданное окружение

доработка примера №1 из лабораторной работы. Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где, a – числитель, b – знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы операции: add, sub, mul, div, equal, операции:

Листинг 1 – Код примера №1

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

class Rational:
    def __init__(self, a: int = 0, b: int = 1):
        if b == 0:
            raise ValueError()
        self.__numerator = abs(int(a))
        self.__denominator = abs(int(b))
        self.__reduce()

    # Сокращение дроби
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a: int, b: int):
            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) -> int:
    return self.__numerator

@property
def denominator(self) -> int:
    return self.__denominator

# Прочитать значение дроби с клавиатуры. Дробь вводится
# как a/b.
def read(self, prompt: str = None):
    line = input(prompt) if prompt else input()
    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) -> str:
    return 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)
    print(f'R1: {r1.display()}')

    r2 = Rational()
    r2.read('Введите обыкновенную дробь: ')
    print(f'R2: {r2.display()}')

    r3 = r2.add(r1)
    print(f'R3 (R2 + R1): {r3.display()}')

    r4 = r2.sub(r1)
    print(f'R4 (R2 - R1): {r4.display()}')

```

```

r5 = r2.mul(r1)
print(f'R5 (R2 * R1): {r5.display()}')

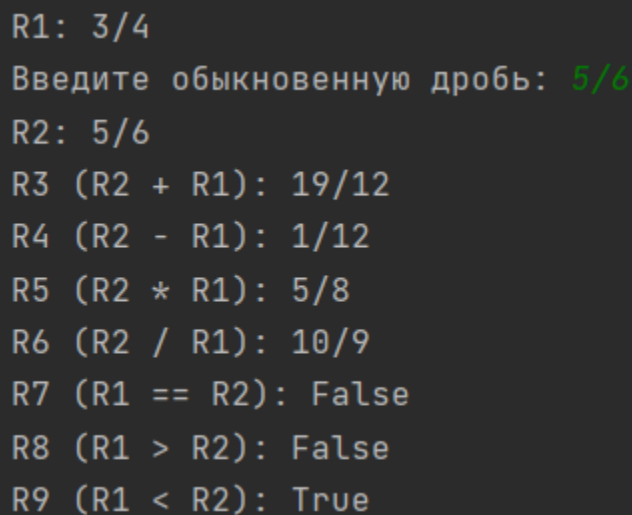
r6 = r2.div(r1)
print(f'R6 (R2 / R1): {r6.display()}')

r7 = r1.equals(r2)
print(f'R7 (R1 == R2): {r7}')

r8 = r1.greater(r2)
print(f'R8 (R1 > R2): {r8}')

r9 = r1.less(r2)
print(f'R9 (R1 < R2): {r9}')

```

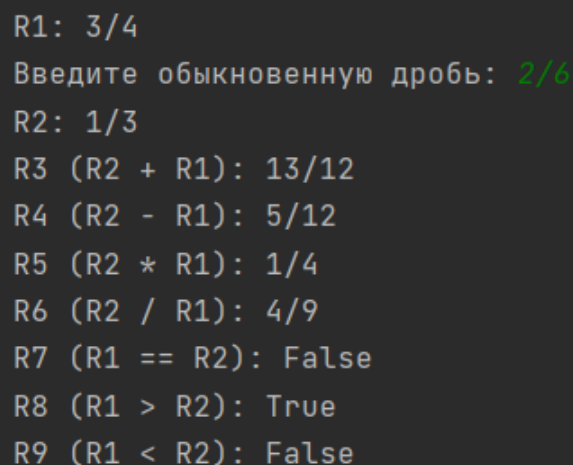


```

R1: 3/4
Введите обыкновенную дробь: 5/6
R2: 5/6
R3 (R2 + R1): 19/12
R4 (R2 - R1): 1/12
R5 (R2 * R1): 5/8
R6 (R2 / R1): 10/9
R7 (R1 == R2): False
R8 (R1 > R2): False
R9 (R1 < R2): True

```

Рисунок 8 – Пример вывода программы (1)



```

R1: 3/4
Введите обыкновенную дробь: 2/6
R2: 1/3
R3 (R2 + R1): 13/12
R4 (R2 - R1): 5/12
R5 (R2 * R1): 1/4
R6 (R2 / R1): 4/9
R7 (R1 == R2): False
R8 (R1 > R2): True
R9 (R1 < R2): False

```

Рисунок 9 – Пример вывода программы (2)

ешение индивидуального задания №1 (вариант №12). 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: «`__init__`», ввод с клавиатуры «`read`» и 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.

12. Поле `first` — дробное число, координата x точки на плоскости; поле `second` — дробное число, координата y точки на плоскости. Реализовать метод `distance()` — расстояние точки от начала координат.

Рисунок 10 – Индивидуальное задание №1 для варианта №12

Листинг 2 – Код индивидуального задания №1

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

import math

def _is_number(value) -> bool:
    """
    Проверяет, можно ли преобразовать значение в число с плавающей запятой.
    :param value: Значение для проверки
    :return: True, если значение является числом, иначе False
    """
    try:
        float(value)
        return True
    except ValueError:
        return False

class Point:
    def __init__(self, first: float = 0.0, second: float = 0.0):
        """
        Инициализирует точку с двумя координатами (first и second).
        :param first: Координата X точки
        :param second: Координата Y точки
        """
        if not _is_number(first) or not _is_number(second):
            raise ValueError("Координаты должны быть числами!")
```

```

        self.first = float(first)
        self.second = float(second)

    def read(self):
        """
        Ввод координат точки с клавиатуры.
        """
        first_input = input('Введите координату X: ')
        if not _is_number(first_input):
            raise ValueError('Координата должна быть числом!')

        second_input = input('Введите координату Y: ')
        if not _is_number(second_input):
            raise ValueError('Координата должна быть числом!')

        self.first = float(first_input)
        self.second = float(second_input)

    def display(self):
        """
        Вывод координат точки на экран.
        """
        print(f'Точка: ({self.first}, {self.second})')

    def distance(self) -> float:
        """
        Вычисляет расстояние от точки до начала координат.
        :return: Расстояние от начала координат до точки
        """
        return math.sqrt(self.first ** 2 + self.second ** 2)

def make_point(first: float, second: float) -> Point:
    """
    Создает объект Point с заданными координатами.
    :param first: Координата X
    :param second: Координата Y
    :return: Объект Point
    :raises ValueError: Если аргументы не являются числами
    """
    if not _is_number(first) or not _is_number(second):
        raise ValueError("Координаты должны быть числами!")
    return Point(first, second)

if __name__ == '__main__':
    # Пример использования класса Point
    try:
        # Создание точки с помощью функции make_point
        point = make_point(3.0, 4.0)
        point.display()

        # Ввод координат с клавиатуры
        point.read()
        point.display()

        # Вычисление расстояния до начала координат
        distance = point.distance()
        print(f'Расстояние от точки до начала координат: {distance:.2f}')
    except ValueError as e:
        print("Ошибка:", e)

```

```
Точка: (3.0, 4.0)
Введите координату X: 6
Введите координату Y: 8
Точка: (6.0, 8.0)
Расстояние от точки до начала координат: 10.00
```

Рисунок 11 – Пример выполнения программы (1)

```
Точка: (3.0, 4.0)
Введите координату X: abc
Ошибка: Координата должна быть числом!
```

Рисунок 12 – Пример выполнения программы (2)

ешение индивидуального задания №2 (вариант №12). Составить программу с использованием классов и объектов для решения задачи. Во всех заданиях обязательно должны присутствовать: «__init__», ввод с клавиатуры «read» и вывод на экран «display»:

12. Создать класс Fraction для работы с дробными числами. Число должно быть представлено двумя целочисленными полями: целая часть и дробная часть. Реализовать арифметические операции сложения, вычитания, умножения и операции сравнения.

Рисунок 13 – Индивидуальное задание №2 для варианта №12

Листинг 3 – Код индивидуального задания №2

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

class Fraction:
    def __init__(self, whole=0, fraction=0):
        """
        Инициализация дроби с целой и дробной частями.

        :param whole: Целая часть
        :param fraction: Дробная часть
        """
        self.whole = int(whole)
```

```

        self.fraction = int(fraction)
        self.normalize()

def normalize(self):
    """
    Нормализация дроби. Приводит дробную часть к допустимому значению.
    """
    if self.fraction >= 10:
        self.whole += self.fraction // 10
        self.fraction %= 10

def read(self):
    """
    Ввод дробного числа с клавиатуры в формате "целая часть.дробная
    часть".
    """
    value = input("Введите дробь в формате 'целая часть.дробная часть':
")
    whole, fraction = map(int, value.split('.'))
    self.whole = whole
    self.fraction = fraction
    self.normalize()

def display(self):
    """
    Вывод дробного числа в формате "целая часть.дробная часть".
    """
    print(f"{self.whole}.{self.fraction}")

def add(self, other):
    """
    Сложение двух дробей.

    :param other: Другая дробь
    :return: Новая дробь — результат сложения
    """
    whole = self.whole + other.whole
    fraction = self.fraction + other.fraction
    return Fraction(whole, fraction)

def sub(self, other):
    """
    Вычитание двух дробей.

    :param other: Другая дробь
    :return: Новая дробь — результат вычитания
    """
    whole = self.whole - other.whole
    fraction = self.fraction - other.fraction
    if fraction < 0:
        whole -= 1
        fraction += 10
    return Fraction(whole, fraction)

def mul(self, other):
    """
    Умножение двух дробей.

    :param other: Другая дробь
    :return: Новая дробь — результат умножения
    """
    whole = (self.whole * 10 + self.fraction) * (other.whole * 10 +
other.fraction) // 10

```

```

        fraction = (self.whole * 10 + self.fraction) * (other.whole * 10 +
other.fraction) % 10
        return Fraction(whole, fraction)

    def equals(self, other):
        """
        Проверка на равенство двух дробей.

        :param other: Другая дробь
        :return: True, если дроби равны, иначе False
        """
        return (self.whole == other.whole) and (self.fraction ==
other.fraction)

    def greater(self, other):
        """
        Проверка, больше ли текущая дробь другой.

        :param other: Другая дробь
        :return: True, если текущая дробь больше, иначе False
        """
        return (self.whole * 10 + self.fraction) > (other.whole * 10 +
other.fraction)

    def less(self, other):
        """
        Проверка, меньше ли текущая дробь другой.

        :param other: Другая дробь
        :return: True, если текущая дробь меньше, иначе False
        """
        return (self.whole * 10 + self.fraction) < (other.whole * 10 +
other.fraction)

def make_fraction(whole, fraction):
    """
    Функция для создания объекта Fraction.

    :param whole: Целая часть
    :param fraction: Дробная часть
    :return: Новый объект Fraction
    """
    if not isinstance(whole, int) or not isinstance(fraction, int):
        print("Ошибка: Аргументы должны быть целыми числами!")
        return None
    return Fraction(whole, fraction)

if __name__ == '__main__':
    f1 = make_fraction(3, 5)
    f1.display()

    f2 = Fraction()
    f2.read()
    f2.display()

    f3 = f1.add(f2)
    print("Сложение:")
    f3.display()

    f4 = f1.sub(f2)
    print("Вычитание:")

```

```

f4.display()

f5 = f1.mul(f2)
print("Умножение:")
f5.display()

print("Равенство:", f1.equals(f2))
print("f1 больше f2:", f1.greater(f2))
print("f1 меньше f2:", f1.less(f2))

```

```

3.5
Введите дробь в формате 'целая часть.дробная часть': 5.8
5.8
Сложение:
9.3
Вычитание:
-3.7
Умножение:
203.0
Равенство: False
f1 больше f2: False
f1 меньше f2: True

```

Рисунок 14 – Пример вывода программы (1)

```

3.5
Введите дробь в формате 'целая часть.дробная часть': 3.5
3.5
Сложение:
7.0
Вычитание:
0.0
Умножение:
122.5
Равенство: True
f1 больше f2: False
f1 меньше f2: False

```

Рисунок 15 – Пример вывода программы (2)

ольем ветки develop и main/master и отправим изменения на удаленный репозиторий:

Ссылка: https://github.com/daxstrong/OOP_1.git

```
(venv) PS C:\Users\Ilya\Documents\labs\00P\00P_1> git log --oneline
e8fdbaa (HEAD -> develop) final changes
38cd0ee (origin/main, origin/HEAD, main) Update README.md
7d34903 Initial commit
```

Рисунок 16 – История коммитов

```
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
(venv) PS C:\Users\Ilya\Documents\labs\00P\00P_1> git merge develop
Updating 38cd0ee..e8fdbaa
Fast-forward
 .idea/.gitignore          |  8 ++
 .idea/00P_1.iml           | 10 ++
 .idea/inspectionProfiles/profiles_settings.xml |  6 +
 .idea/misc.xml            |  4 +
 .idea/modules.xml         |  8 ++
 .idea/vcs.xml             |  6 +
 Individual_1.py           | 91 +++++
 example_1.py              | 148 +++++
 individual_2.py           | 141 +++++
 9 files changed, 422 insertions(+)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/00P_1.iml
 create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/modules.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 Individual_1.py
 create mode 100644 example_1.py
 create mode 100644 individual_2.py
(venv) PS C:\Users\Ilya\Documents\labs\00P\00P_1>
```

Рисунок 17 – Слияние веток main и develop

```
(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_1> git push origin main
info: please complete authentication in your browser...
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 12 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 5.00 KiB | 853.00 KiB/s, done.
Total 13 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/daxstrong/OOP\_1.git
 38cd0ee..e8fdbaa main -> main
(venv) PS C:\Users\Ilya\Documents\labs\OOP\OOP_1>
```

Рисунок 18 – Отправка изменений на удаленный репозиторий

Вывод: приобрели полезные навыки, связанные с работой с классами и их экземплярами – объектами. Решили задачи при помощи новых знаний о возможностях ООП в Python. При написании программ использовался язык программирования Python версии 3.10.

Ответы на контрольные вопросы:

как осуществляется объявление класса в языке Python?

Для объявления класса в Python используется ключевое слово `class`, за которым идет имя класса с двоеточием. Внутри класса обычно размещаются его методы и атрибуты. Принято давать именам классов в верхнем регистре, чтобы их легко отличить от переменных и функций.

чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибуты класса принадлежат самому классу и разделяются всеми экземплярами этого класса. В отличие от них, атрибуты экземпляра создаются в методах класса, например, в конструкторе `__init__()`, и уникальны для каждого объекта, то есть их значения могут различаться у разных экземпляров. каково назначение методов класса?

Методы класса — это функции, которые определены внутри класса и работают с его атрибутами. Они описывают поведение объектов данного класса, взаимодействуют с их состоянием и помогают организовать код. Методы способствуют удобному структурированию и повторному использованию логики.

для чего предназначен метод `__init__()` класса?

Метод `__init__()` в Python выполняет роль конструктора и автоматически вызывается при создании нового экземпляра класса. Он отвечает за инициализацию атрибутов объекта, присваивая им значения, которые могут быть переданы при создании объекта. Это позволяет настроить начальное состояние объекта.

каково назначение `self`?

— это ссылка на текущий экземпляр класса, она используется для доступа к его атрибутам и методам. Параметр `self` помогает различать атрибуты класса и экземпляра, а также позволяет методам работать с конкретным объектом, вызвавшим этот метод.

как добавить атрибуты в класс?

Атрибуты можно добавить в класс, прописав их непосредственно в теле класса или в методе `__init__()` для каждого экземпляра. Кроме того, можно добавлять атрибуты динамически, используя `self` для привязки новых переменных к объекту после его создания.

ак осуществляется управление доступом к методам и атрибутам в языке

не имеет строгих механизмов доступа к атрибутам и методам, но использует соглашения по именованию. Например, атрибуты, начинающиеся с одного или двух подчеркиваний, предполагаются как защищенные или приватные, и их не рекомендуется использовать за пределами класса. Однако это лишь рекомендация, и доступ к таким атрибутам технически возможен.

аково назначение функции `isinstance()`?

Функция `isinstance()` проверяет, является ли объект экземпляром указанного класса или его подкласса. Это полезно для того, чтобы убедиться в правильности типа объекта перед его использованием в функции или методе, предотвращая ошибки при обработке данных.