

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Перегрузка операторов в языке Python»**

**Отчет по лабораторной работе № 4.2(14)
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

Подпись студента _____

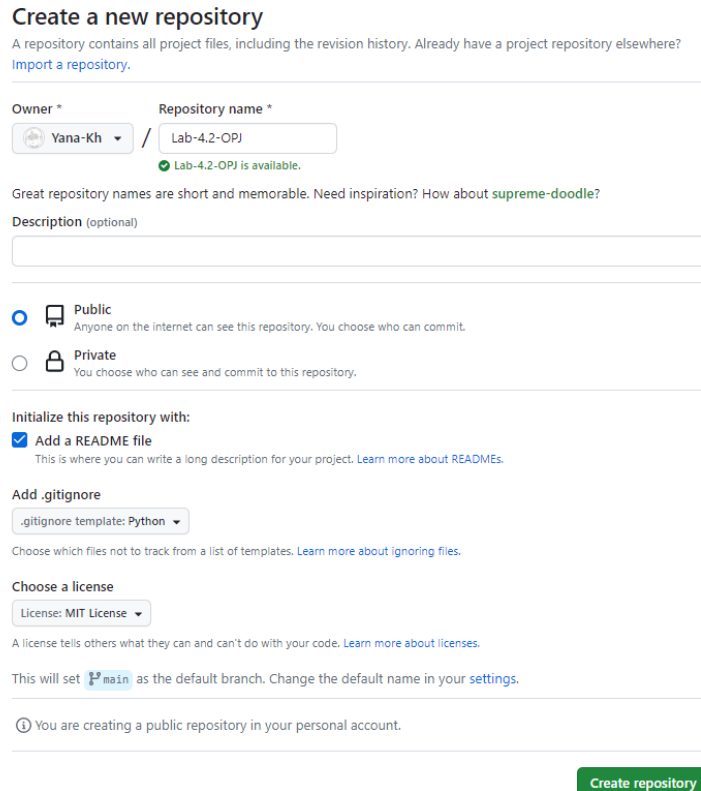
Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

Цель работы: приобретение навыков по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT и язык программирования Python.



Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * Yana-Kh / Repository name * Lab-4.2-OPJ
Lab-4.2-OPJ is available.

Great repository names are short and memorable. Need inspiration? How about [supreme-doodle?](#)

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
.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

3. Выполните клонирование созданного репозитория.

```
Microsoft Windows [Version 10.0.19045.2846]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\антон>cd C:\Users\антон\Desktop\Git

C:\Users\антон\Desktop\Git>git clone https://github.com/Yana-Kh/Lab-4.2-OPJ.git
Cloning into 'Lab-4.2-OPJ'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\антон\Desktop\Git>
```

Рисунок 2 – Клонирование репозитория

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

```
C:\Users\anton\Desktop\Git\Lab-4.2-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
        new file:   requirements.txt

C:\Users\anton\Desktop\Git\Lab-4.2-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

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

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
```

Рисунок 4 – Организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

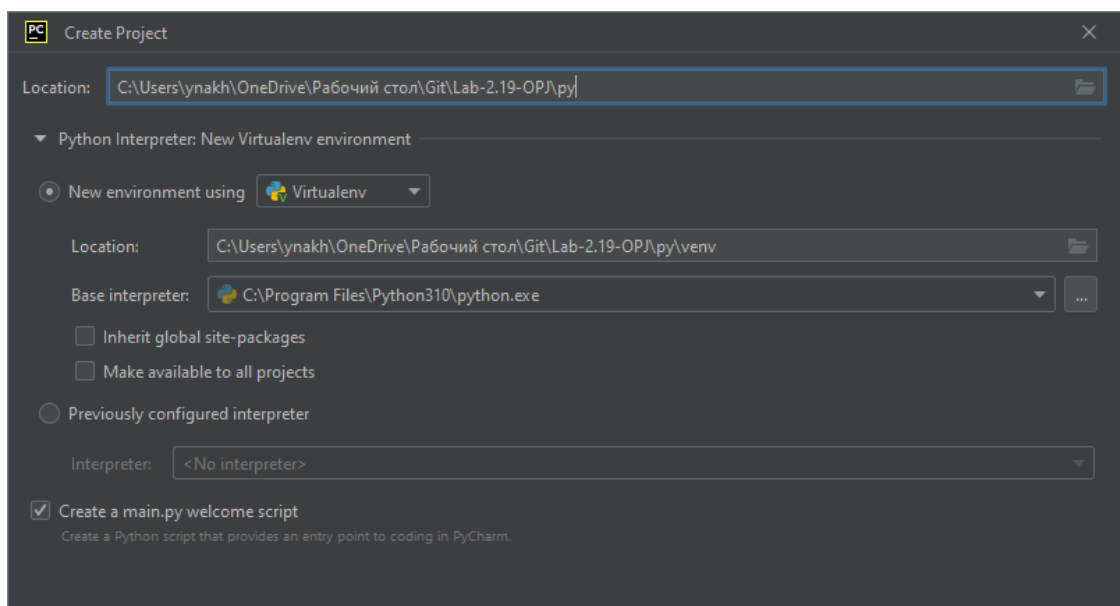


Рисунок 5 – Создание проекта PyCharm

7. Проработать примеры лабораторной работы.

Пример 1.

Код:

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

import math

class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Vector2D({}, {})'.format(self.x, self.y)

    def __str__(self):
        return '({}, {})'.format(self.x, self.y)

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __iadd__(self, other):
        self.x += other.x
        self.y += other.y
        return self

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __isub__(self, other):
        self.x -= other.x
        self.y -= other.y
        return self

    def __abs__(self):
        return math.hypot(self.x, self.y)

    def __bool__(self):
        return self.x != 0 or self.y != 0

    def __neg__(self):
        return Vector2D(-self.x, -self.y)

if __name__ == '__main__':
    x = Vector2D(3, 4)
    print(x)
    print(abs(x))
    y = Vector2D(5, 6)
    print(y)
    print(x + y)
    print(x - y)
    print(-x)
    x += y
    print(x)
    print(bool(x))
    z = Vector2D(0, 0)
```

```
print(bool(z))
print(-z)
```

```
0.03333333333333333
(3, 4)
5.0
(5, 6)
(8, 10)
(-2, -2)
(-3, -4)
(8, 10)
True
False
(0, 0)
```

Рисунок 6 – Результат работы программы

Пример 2.

Код:

```
#!/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("Illegal value of the denominator")
        self.__numerator = a
        self.__denominator = 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)

        sign = 1
        if (self.__numerator > 0 > self.__denominator) or \
            (self.__numerator < 0 < self.__denominator):
            sign = -1

        a, b = abs(self.__numerator), abs(self.__denominator)
        c = gcd(a, b)
        self.__numerator = sign * (a // c)
```

```

        self.__denominator = b // c

# Клонировать дробь.
def __clone(self):
    return Rational(self.__numerator, self.__denominator)

@property
def numerator(self):
    return self.__numerator

@numerator.setter
def numerator(self, value):
    self.__numerator = int(value)
    self.__reduce()

@property
def denominator(self):
    return self.__denominator

@denominator.setter
def denominator(self, value):
    value = int(value)
    if value == 0:
        raise ValueError("Illegal value of the denominator")
    self.__denominator = value
    self.__reduce()

# Привести дробь к строке.
def __str__(self):
    return f"{self.__numerator} / {self.__denominator}"

def __repr__(self):
    return self.__str__()

# Привести дробь к вещественному значению.
def __float__(self):
    return self.__numerator / self.__denominator

# Привести дробь к логическому значению.
def __bool__(self):
    return self.__numerator != 0

# Сложение обыкновенных дробей.
def __iadd__(self, rhs): # +=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator + \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
        self.__numerator, self.__denominator = a, b
        self.__reduce()
        return self
    else:
        raise ValueError("Illegal type of the argument")

def __add__(self, rhs): # +
    return self.__clone().__iadd__(rhs)

# Вычитание обыкновенных дробей.
def __isub__(self, rhs): # -=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator - \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
        self.__numerator, self.__denominator = a, b

```

```

        self.__reduce()
        return self
    else:
        raise ValueError("Illegal type of the argument")

def __sub__(self, rhs): # -
    return self.__clone().__isub__(rhs)

# Умножение обыкновенных дробей.
def __imul__(self, rhs): # *=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.numerator
        b = self.denominator * rhs.denominator
        self.__numerator, self.__denominator = a, b
        self.__reduce()
        return self
    else:
        raise ValueError("Illegal type of the argument")

def __mul__(self, rhs): # *
    return self.__clone().__imul__(rhs)

# Деление обыкновенных дробей.
def __itruediv__(self, rhs): # /=
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator
        b = self.denominator * rhs.numerator
        if b == 0:
            raise ValueError("Illegal value of the denominator")
        self.__numerator, self.__denominator = a, b
        self.__reduce()
        return self
    else:
        raise ValueError("Illegal type of the argument")

def __truediv__(self, rhs): # /
    return self.__clone().__itruediv__(rhs)

# Отношение обыкновенных дробей.
def __eq__(self, rhs): # ==
    if isinstance(rhs, Rational):
        return (self.numerator == rhs.numerator) and \
            (self.denominator == rhs.denominator)
    else:
        return False

def __ne__(self, rhs): # !=
    if isinstance(rhs, Rational):
        return not self.__eq__(rhs)
    else:
        return False

def __gt__(self, rhs): # >
    if isinstance(rhs, Rational):
        return self.__float__() > rhs.__float__()
    else:
        return False

def __lt__(self, rhs): # <
    if isinstance(rhs, Rational):
        return self.__float__() < rhs.__float__()
    else:
        return False

```

```

def __ge__(self, rhs): # >=
    if isinstance(rhs, Rational):
        return not self.__lt__(rhs)
    else:
        return False

def __le__(self, rhs): # <=
    if isinstance(rhs, Rational):
        return not self.__gt__(rhs)
    else:
        return False

if __name__ == '__main__':
    r1 = Rational(3, 4)
    print(f"r1 = {r1}")
    r2 = Rational(5, 6)
    print(f"r2 = {r2}")
    print(f"r1 + r2 = {r1 + r2}")
    print(f"r1 - r2 = {r1 - r2}")
    print(f"r1 * r2 = {r1 * r2}")
    print(f"r1 / r2 = {r1 / r2}")
    print(f"r1 == r2: {r1 == r2}")
    print(f"r1 != r2: {r1 != r2}")
    print(f"r1 > r2: {r1 > r2}")
    print(f"r1 < r2: {r1 < r2}")
    print(f"r1 >= r2: {r1 >= r2}")
    print(f"r1 <= r2: {r1 <= r2}")

```

```

r1 = 3 / 4
r2 = 5 / 6
r1 + r2 = 19 / 12
r1 - r2 = -1 / 12
r1 * r2 = 5 / 8
r1 / r2 = 9 / 10
r1 == r2: False
r1 != r2: True
r1 > r2: False
r1 < r2: True
r1 >= r2: False
r1 <= r2: True

```

Process finished with exit code 0

Рисунок 7 – Результат работы программы

8. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

Задание 1. Выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально задействовав имеющиеся в Python средства перегрузки операторов.

Код:

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

"""
Выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально
задействовав
имеющиеся в Python средства перегрузки операторов.
Вариант 29(9)
9. Поле first – целое положительное число, часы; поле second – целое
положительное
число, минуты. Реализовать метод minutes() – приведение времени в минуты.
"""

class MyTime:
    def __init__(self, first=0, second=0):
        if isinstance(first, int) and isinstance(second, int) and first >= 0
and second >= 0:
            self.first = first
            self.second = second
        else:
            raise ValueError

    def __str__(self):
        return f"First (hours): {self.first}\nSecond (minutes):
{self.second}"

    def __add__(self, other):
        if isinstance(other, MyTime):
            total_minutes = self.first * 60 + self.second + other.first * 60
+ other.second
            hours = total_minutes // 60
            minutes = total_minutes % 60
            return MyTime(hours, minutes)
        else:
            raise TypeError("Unsupported operand type for +")

    def __sub__(self, other):
        if isinstance(other, MyTime):
            total_minutes = self.first * 60 + self.second - (other.first * 60
+ other.second)
            if total_minutes >= 0:
                hours = total_minutes // 60
                minutes = total_minutes % 60
                return MyTime(hours, minutes)
            else:
                raise ValueError("Resulting time cannot be negative")
        else:
            raise TypeError("Unsupported operand type for -")

    def read(self):
        try:
            self.first = int(input("Enter hours: "))
            self.second = int(input("Enter minutes: "))
        except:
```

```

        print("Error")

    def display(self):
        print(f"First (hours): {self.first}")
        print(f"Second (minutes): {self.second}")

    def minutes(self):
        print(f"Time in minutes: {self.first * 60 + self.second}")

def make_MyTime(first, second):
    return MyTime(first, second)

if __name__ == '__main__':
    time1 = MyTime(3, 45)
    time1.display()
    time1.minutes()
    time2 = MyTime(1, 30)
    time3 = time1 + time2
    time3.display()
    time4 = time1 - time2
    time4.display()

```

```

First (hours): 3
Second (minutes): 45
Time in minutes: 225
First (hours): 5
Second (minutes): 15
First (hours): 2
Second (minutes): 15

Process finished with exit code 0

```

Рисунок 8 – Результат работы программы

Задание 2. Дополнительно к требуемым в заданиях операциям перегрузить операцию индексирования []. Максимально возможный размер списка задать константой. В отдельном поле size должно храниться максимальное для данного объекта количество элементов списка; реализовать метод size(), возвращающий установленную длину. Если количество элементов списка изменяется во время работы, определить в классе поле count. Первоначальные значения size и count устанавливаются конструктором. В тех задачах, где возможно, реализовать конструктор инициализации строкой.

Вариант 29 (8)

8. Реализовать класс Money, используя для представления суммы денег список словарей. Словарь имеет два ключа: номинал купюры и количество

купюр данного достоинства. Номиналы представить как строку. Элемент списка словарей с меньшим индексом содержит меньший номинал.

Код:

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

"""
Выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально
задействовав имеющиеся в Python средства перегрузки операторов.
Вариант 29(9)
9. Поле first – целое положительное число, часы; поле second – целое
положительное число, минуты. Реализовать метод minutes() – приведение времени в минуты.
"""

class MyTime:
    def __init__(self, first=0, second=0):
        if isinstance(first, int) and isinstance(second, int) and first >= 0
and second >= 0:
            self.first = first
            self.second = second
        else:
            raise ValueError

    def __str__(self):
        return f"First (hours): {self.first}\nSecond (minutes):
{self.second}"

    def __add__(self, other):
        if isinstance(other, MyTime):
            total_minutes = self.first * 60 + self.second + other.first * 60
+ other.second
            hours = total_minutes // 60
            minutes = total_minutes % 60
            return MyTime(hours, minutes)
        else:
            raise TypeError("Unsupported operand type for +")

    def __sub__(self, other):
        if isinstance(other, MyTime):
            total_minutes = self.first * 60 + self.second - (other.first * 60
+ other.second)
            if total_minutes >= 0:
                hours = total_minutes // 60
                minutes = total_minutes % 60
                return MyTime(hours, minutes)
            else:
                raise ValueError("Resulting time cannot be negative")
        else:
            raise TypeError("Unsupported operand type for -")

    def read(self):
        try:
            self.first = int(input("Enter hours: "))
            self.second = int(input("Enter minutes: "))
        except:
            print("Error")
```

```

def display(self):
    print(f"First (hours): {self.first}")
    print(f"Second (minutes): {self.second}")

def minutes(self):
    print(f"Time in minutes: {self.first * 60 + self.second}")

def make_MyTime(first, second):
    return MyTime(first, second)

if __name__ == '__main__':
    time1 = MyTime(3, 45)
    time1.display()
    time1.minutes()
    time2 = MyTime(1, 30)
    time3 = time1 + time2
    time3.display()
    time4 = time1 - time2
    time4.display()

```

```

Размер словаря: 4
None
Общая сумма: 1650
None
Размер словаря: 9
None
Общая сумма: 26650
None
Размер словаря: 9
None
Общая сумма: 26650
None
{'номинал': '10', 'количество': 20}
{'номинал': '200', 'количество': 1}

```

Рисунок 9 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.





 ex1.py	ex
 ex2.py	ex
 id1.py	id
 id2.py	id

Рисунок 10 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

1. Какие средства существуют в Python для перегрузки операций?

Перегрузка операторов – один из способов реализации полиморфизма, когда мы можем задать свою реализацию какого-либо метода в своём классе.

Например:

```
class A:
    def go(self):
        print('Go, A!')

class B(A):
    def go(self, name):
        print('Go, {}'.format(name))
```

В данном примере класс B наследует класс A, но переопределяет метод go, поэтому он имеет мало общего с аналогичным методом класса A. Однако в python имеются методы, которые, как правило, не вызываются напрямую, а вызываются встроенными функциями или операторами. Например, метод `__init__` перегружает конструктор класса. Конструктор - создание экземпляра класса

2. Какие существуют методы для перегрузки арифметических операций и операций отношения в языке Python?

`__add__(self, other)` - сложение. `x + y` вызывает `x.__add__(y)`.
`__sub__(self, other)` - вычитание (`x - y`).
`__mul__(self, other)` - умножение (`x * y`).
`__truediv__(self, other)` - деление (`x / y`).
`__floordiv__(self, other)` - целочисленное деление (`x // y`).
`__mod__(self, other)` - остаток от деления (`x % y`).
`__divmod__(self, other)` - частное и остаток (`divmod(x, y)`).
`__pow__(self, other[, modulo])` - возведение в степень (`x ** y`, `pow(x, y[, modulo])`).
`__lshift__(self, other)` - битовый сдвиг влево (`x << y`).
`__rshift__(self, other)` - битовый сдвиг вправо (`x >> y`).
`__and__(self, other)` - битовое И (`x & y`).
`__xor__(self, other)` - битовое ИСКЛЮЧАЮЩЕЕ ИЛИ (`x ^ y`).
`__or__(self, other)` - битовое ИЛИ (`x | y`).

3. В каких случаях будут вызваны следующие методы: `__add__` , `__iadd__` и `__radd__` ? Приведите примеры.

Операция `x + y` будет сначала пытаться вызвать `x.__add__(y)` и только в том случае, если это не получилось, будет пытаться вызвать `y.__radd__(x)`.

`__iadd__(self, other)` - +=

4. Для каких целей предназначен метод `__new__` ? Чем он отличается от метода `__init__` ?

`__new__(cls[, ...])` — управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращать экземпляр класса для его последующей его передачи методу `__init__` .

5. Чем отличаются методы `__str__` и `__repr__` ?

`__repr__(self)` - вызывается встроенной функцией `repr`; возвращает "сырые" данные, использующиеся для внутреннего представления в python.

`__str__(self)` - вызывается функциями `str`, `print` и `format`. Возвращает строковое представление объекта