

**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
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Трушева В. О. _____. « » 2023г.

Подпись студента _____

Работа защищена « _____ 20__ г.

Проверила Воронкин Р.А. _____

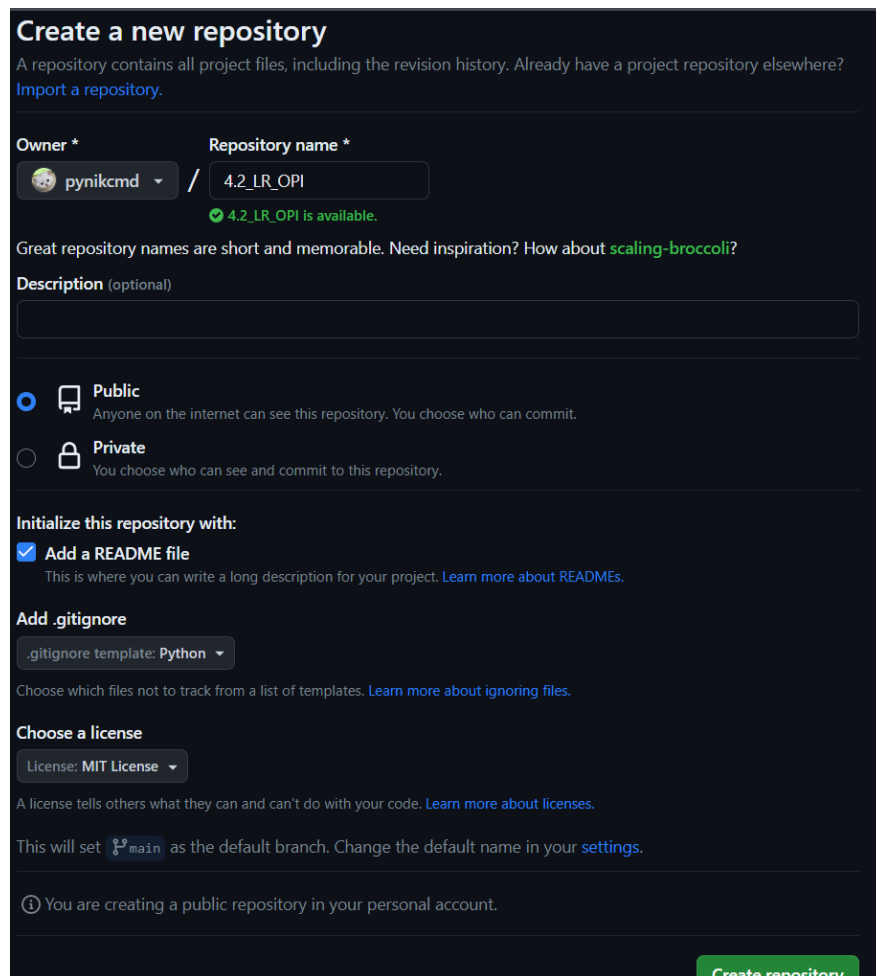
(подпись)

Ставрополь 2023

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

Методика и порядок выполнения работы

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




The screenshot shows the GitHub 'Create a new repository' interface. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two main input fields: 'Owner' (set to 'pynikcmd') and 'Repository name' (set to '4.2_LR_OPI'). A green checkmark indicates that the name is available. There is a suggestion for 'scaling-broccoli'. Below the name field is a 'Description (optional)' text area. The 'Visibility' section has two options: 'Public' (selected) and 'Private'. The 'Initialize this repository with:' section has a checked box for 'Add a README file'. Below this is the 'Add .gitignore' section with a dropdown menu set to 'Python'. The 'Choose a license' section has a dropdown menu set to 'MIT License'. At the bottom, there is a green 'Create repository' button. A small note at the bottom left states: 'You are creating a public repository in your personal account.'

3. Выполните клонирование созданного репозитория.

```
D:\fgit>git clone https://github.com/pynikcmd/4.2_LR_OPI.git
Cloning into '4.2_LR_OPI'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 5.12 KiB | 5.12 MiB/s, done.
```

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

 .gitignore Update .gitignore

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

```
D:\fgit\4.2_LR_OPI>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]

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? []
Hooks and filters directory? [D:/fgit/4.2_LR_OPI/.git/hooks]
```

6. Создайте проект PyCharm в папке репозитория.

7. Проработайте примеры лабораторной работы.

```
44  
45 ▶ if __name__ == '__main__':  
46     x = Vector2D(3, 4)  
47     print(x)  
48     print(abs(x))  
49     y = Vector2D(5, 6)  
50     print(y)  
51     print(x + y)  
52     print(x - y)  
53     print(-x)  
54     x += y  
55     print(x)  
56     print(bool(x))  
57     z = Vector2D(0, 0)  
58     print(bool(z))  
59     print(-z)  
  
if __name__ == '__main__':  
Primer_1 x  
D:\fgit\4.2_LR_OPI\Tasks\venv\Scripts\  
↑ (3, 4)  
↓ 5.0  
↺ (5, 6)  
↻ (8, 10)  
🖨 (-2, -2)  
🗑 (-3, -4)  
(8, 10)  
True  
False  
(0, 0)
```

Пример 1

```
5 class Rational:
6     def __init__(self, a=0, b=1):
7         a = int(a)
8         b = int(b)
9         if b == 0:
10             raise ValueError("Illegal value of the denominator")
11         self.__numerator = a
12         self.__denominator = b
13         self.__reduce()
14
15     # Сокращение дроби.
16     def __reduce(self):
17         # Функция для нахождения наибольшего общего делителя
18         def gcd(a, b):
19             if a == 0:
20                 return b
21             elif b == 0:
22                 return a
23             else:
24                 return gcd(b, a % b)
25
26         g = gcd(self.__numerator, self.__denominator)
27         self.__numerator //= g
28         self.__denominator //= g
29
30 if __name__ == '__main__':
31     r1 = Rational(3, 4)
32     r2 = Rational(5, 6)
33     print(r1 + r2)
34     print(r1 - r2)
35     print(r1 * r2)
36     print(r1 / r2)
37     print(r1 == r2)
38     print(r1 != r2)
39     print(r1 > r2)
40     print(r1 < r2)
41     print(r1 >= r2)
42     print(r1 <= r2)
```

Run: Primer_2 x

D:\fgit\4.2_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\4.2_LR_OPI\Ta

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

Пример 2

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.

Вариант – 7 (27)

Условие. Поле `first` — дробное число, левая граница диапазона; поле `second` — дробное число, правая граница диапазона. Реализовать метод `rangecheck()` — проверку заданного числа на принадлежность диапазону.

```
11
12 class Numb:
13     def __init__(self, first, second):
14         self.first = float(first)
15         self.second = float(second)
16
17     def __str__(self):
18         return f"Pair: {self.first}, {self.second}"
19
20     def __contains__(self, number):
21         return self.first <= number <= self.second
22
23     def read(self):
24         self.first = float(input("Введите первое значение: "))
25         self.second = float(input("Введите второе значение: "))
26
27     def display(self):
28         print(self)
29
Numb > __str__
Run: Ind_1 (1) x
D:\fgit\4.2_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\4.2_LR_OPI\Tasks\Ind_1.py
Pair: 1.5, 3.7
Pair: 2.0, 4.5
Введите число для проверки диапазона: 5
5.0 вне диапазона.
Process finished with exit code 0
```

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 строк.

Вариант – 6 (27)

Условие. Создать класс `Fraction` для работы с беззнаковыми дробными десятичными числами. Число должно быть представлено двумя

списками типа `int`: целая и дробная часть, каждый элемент — десятичная цифра. Для целой части младшая цифра имеет меньший индекс, для дробной части старшая цифра имеет меньший индекс (десятые — в нулевом элементе, сотые — в первом, и т. д.). Реальный размер списков задается как аргумент конструктора инициализации. Реализовать арифметические операции сложения, вычитания и умножения, и операции сравнения.

`__getitem__`: Перегрузка операции индексирования `[]` для доступа к элементам списка.

`__setitem__`: Перегрузка операции присваивания `[]` для установки значения элемента списка.

`__str__`: Перегрузка операции преобразования объекта в строку для вывода.

`__add__`: Перегрузка операции сложения `+` для объектов класса `Fraction`.

`__sub__`: Перегрузка операции вычитания `-` для объектов класса `Fraction`.

`__mul__`: Перегрузка операции умножения `*` для объектов класса `Fraction`.

`__eq__`: Перегрузка операции равенства `==` для объектов класса `Fraction`.

`__ne__`: Перегрузка операции неравенства `!=` для объектов класса `Fraction`.

`__lt__`: Перегрузка операции меньше `<` для объектов класса `Fraction`.

`__le__`: Перегрузка операции меньше или равно `<=` для объектов класса `Fraction`.

`__gt__`: Перегрузка операции больше `>` для объектов класса `Fraction`.

`__ge__`: Перегрузка операции больше или равно \geq для объектов класса Fraction.

```
15 class Fraction:
16     def __init__(self, size):
17         self.size = size
18         self.integer_part = [0] * size # Целая часть дроби
19         self.fractional_part = [0] * size # Дробная часть дроби
20         self.count = 0 # Текущее количество элементов в списке
21
22     def size(self):
23         return self.size
24
25     def __getitem__(self, index):
26         # Перегрузка операции индексирования для доступа к элементам списка
27         if 0 <= index < self.size:
28             return (self.integer_part[::-1] + self.fractional_part)[index]
29         else:
30             raise IndexError("Индекс вне допустимого диапазона")
31
32     def __setitem__(self, index, value):
33         # Перегрузка операции присваивания для установки значения элемента списка
34         if 0 <= index < self.size:
35             if 0 <= value <= 9:
36                 if index < self.size // 2:
37                     self.integer_part[self.size - 1 - index] = value
38                 else:
39                     self.fractional_part[index - self.size // 2] = value
40
41 if __name__ == '__main__':
42     run: Ind_2
43     D:\fgit\4.2_LR_OPI\Tasks\venv\Scripts\python.exe D:\fgit\4.2_LR_OPI\Tasks\Ind_2.py
44     12000.34500
45     98000.76000
```

9. Зафиксируйте сделанные изменения в репозитории.

10. Выполните слияние ветки для разработки с веткой main / master.

11. Отправьте сделанные изменения на сервер GitHub.

Контрольные вопросы

1. Какие средства существуют в Python для перегрузки операций?

Заключение оператора в двойное подчёркивание «`__`» с обеих сторон.

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 ($x \wedge y$).

`__or__(self, other)` - битовое ИЛИ ($x | y$).

`__radd__(self, other)`,

`__rsub__(self, other)`,

`__rmul__(self, other)`,

`__rtruediv__(self, other)`,

`__rfloordiv__(self, other)`,

`__rmod__(self, other)`,

`__rdivmod__(self, other)`,

`__rpow__(self, other)`,

`__rlshift__(self, other)`,

`__rrshift__(self, other)`,

`__rand__(self, other)`,

`__rxor__(self, other)`,

`__ror__(self, other)` - делают то же самое, что и арифметические операторы, перечисленные выше, но для аргументов, находящихся справа, и только в случае, если для левого операнда не определён соответствующий метод.

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

Например, 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)` . Аналогично для остальных методов.

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

Он 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__` ?

`__str__(self)` - вызывается функциями `str`, `print` и `format`. Возвращает строковое представление объекта. `__repr__(self)` - вызывается встроенной функцией `repr`; возвращает "сырые" данные, используемые для внутреннего представления в python.