МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 4.2

«Перегрузка операторов в языке Python»

по дисциплине «Основы программной инженерии»

Выполнил студент группы
ПИЖ-б-о-21-1
Зиберов Александр
« » мая 2023 г.
Подпись студента_____
Работа защищена
« »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь, 2023

Цель работы:

Приобретение навыков по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: https://github.com/afk552/lab4.2

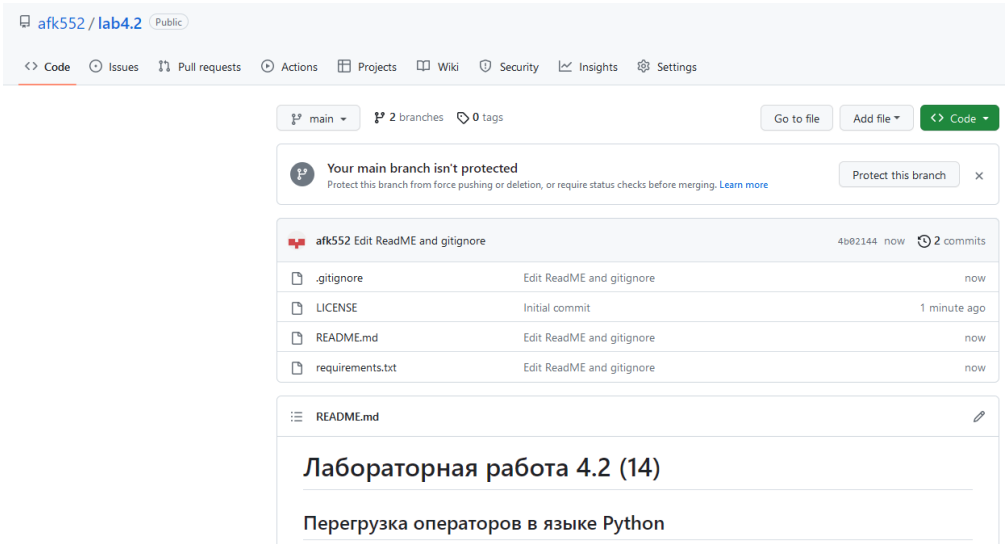


Рисунок 1 – Удаленный репозиторий на GitHub

Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.

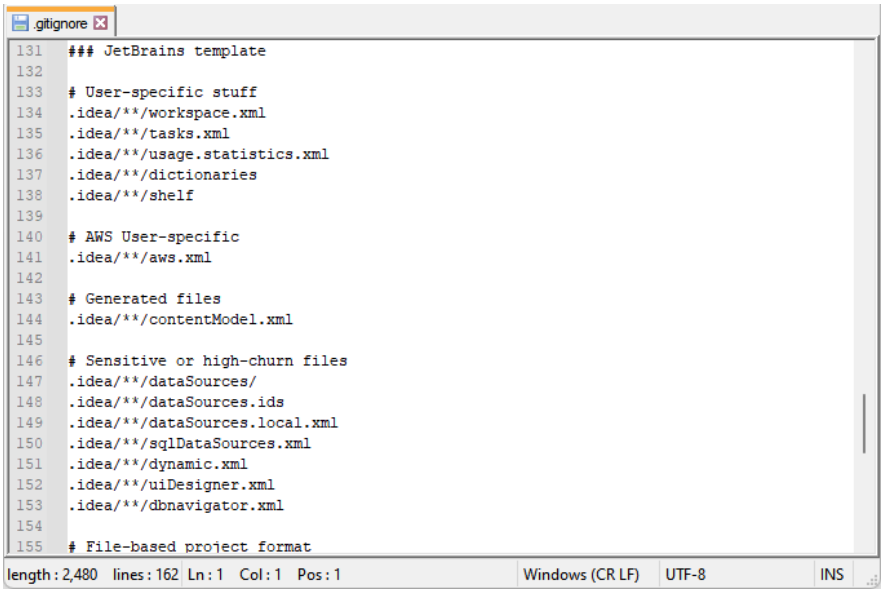


Рисунок 2 – Окно блокнота

Организуйте свой репозиторий в соответствие с моделью ветвления git-flow, рисунок 3.

```
C:\git\lab4.2>git checkout -b develop
Switched to a new branch 'develop'

C:\git\lab4.2>
```

Рисунок 3 – Окно командной строки

Примеры

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


import math


6 usages
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)
```

```
C:\git\lab4.2\venv\Scripts\python.exe C:\git\lab4.2\examples\ex1.py
(3, 4)
5.0
(5, 6)
(8, 10)
(-2, -2)
```

Рисунок 4 – Пример 1

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



13 usages
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.
    7 usages
    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.
    4 usages
    def __clone(self):
        return Rational(self.__numerator, self.__denominator)

    11 usages
    @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:
```

Рисунок 5 – Пример 2 (1)

```
            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}")
```

Рисунок 6 – Пример 2 (2)

Индивидуальное задание.

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

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

```
45      # Перегрузка, вывод интервала
46      def __str__(self):
47          return f"[{self.first}, {self.second}]"
48
49      # Перегрузка, число в интервале
50      def __contains__(self, number):
51          return self.first <= number < self.second
52
        1 usage
53      def rangecheck(self, number=None):
54          if self.first > self.second:
55              raise ValueError("Введенная левая граница больше правой!")
56          if isinstance(number, float):
57              pass
58          else:
59              try:
60                  number = float(input("Задайте число: "))
61              except ValueError:
62                  ValueError("Введено не число!")
63          if self.first <= float(number) <= self.second:
64              print(f"Число {number} в диапазоне [{self.first}, {self.second}]")
65          else:
66              print(f"Число {number} не в [{self.first}, {self.second}]")
67
68
69  if __name__ == "__main__":
70      test = Number()
71      test.read()
72      # Проверка перегрузок
73      test.display()
74      print(test)
75      test.rangecheck(25)
76      print(5 in test)
77
```

Рисунок 7 – Код программы индивидуального задания 1

```
C:\git\lab4.2\venv\Scripts\python.exe C:\git\lab4.2\indiv_task\task1.py
Введите левую границу диапазона: 19.4
Введите правую границу диапазона: 45.8
1: 19.4
2: 45.8
[19.4, 45.8]
Число 25 в диапазоне [19.4, 45.8]
True

Process finished with exit code 0
```

Рисунок 8 – Результат выполнения программы индивидуального задания 1

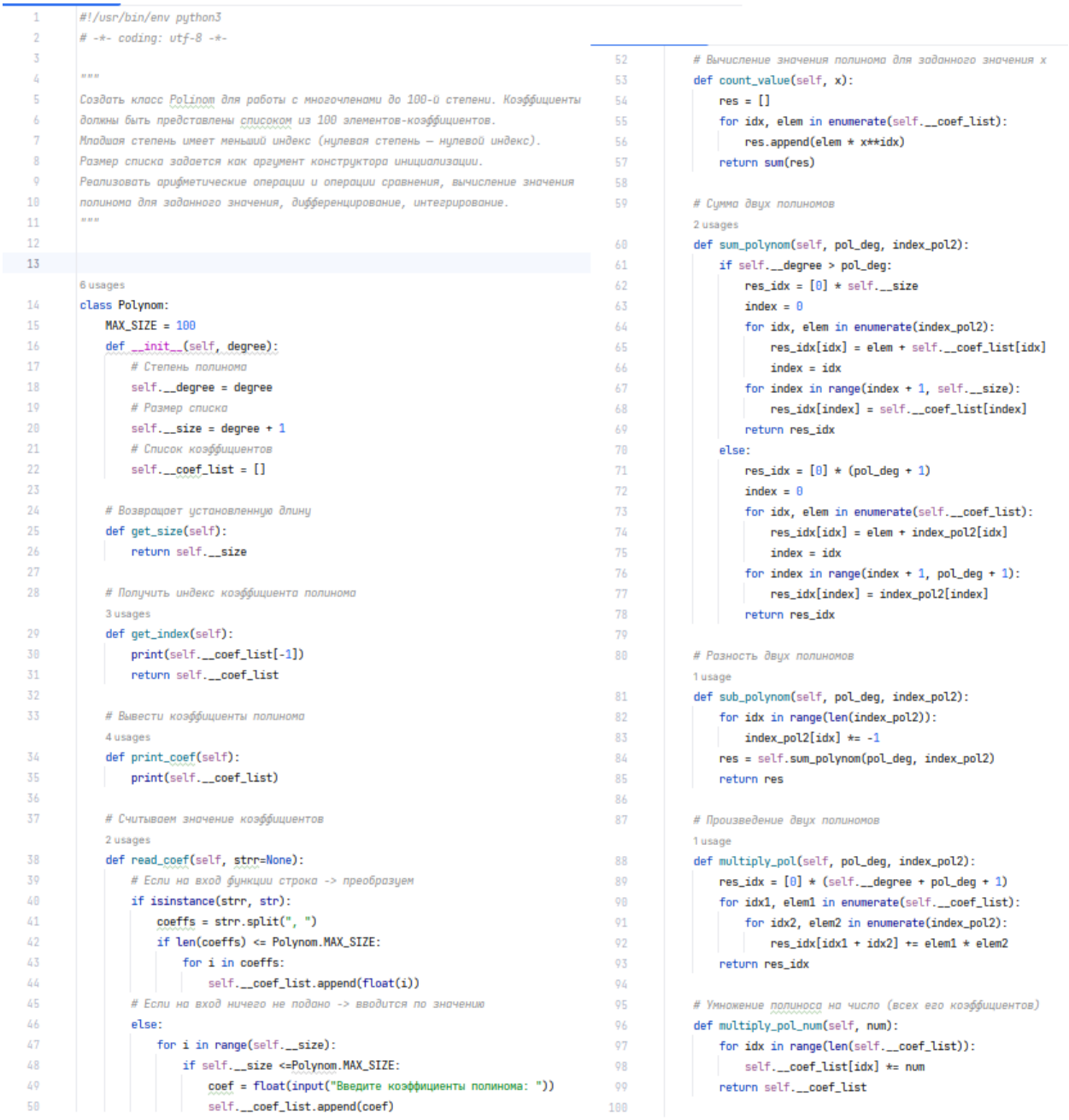
Задание 2. Дополнительно к требуемым в заданиях операциям перегрузить операцию индексирования []. Максимально возможный размер списка задать константой. В отдельном поле size должно храниться максимальное для данного объекта количество элементов списка; реализовать метод size(), возвращающий установленную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

Создать класс Polinom для работы с многочленами до 100-й степени. Коэффициенты должны быть представлены списоком из 100 элементов-коэффициентов. Младшая степень имеет меньший индекс (нулевая степень — нулевой индекс). Размер списка задается как аргумент конструктора инициализации. Реализовать арифметические операции и операции сравнения, вычисление значения полинома для заданного значения , дифференцирование, интегрирование.

```
C:\git\lab4.2\venv\Scripts\python.exe C:\git\lab4.2\indiv_task\task2.py
[]
Введите коэффициенты полинома: 3
Введите коэффициенты полинома: 4
Введите коэффициенты полинома: 2
Введите коэффициенты полинома: 3
[3.0, 4.0, 2.0, 3.0]
[]
[1.0, 3.0, 4.0, 4.0]
4.0
Сумма: [4.0, 7.0, 6.0, 7.0]
4.0
Разность: [2.0, 1.0, -2.0, -1.0]
-4.0
Произведение: [-3.0, -13.0, -26.0, -37.0, -33.0, -20.0, -12.0]
pol1 == pol2: False
pol1 > pol2: False
pol1 < pol2: True
pol1 != pol2: True
Дифференцирование полинома 1: [4.0, 4.0, 9.0]
Интегрирование полинома 1: [0, 3.0, 2.0, 0.6666666666666666, 0.75]

Process finished with exit code 0
```

Рисунок 9 – Результат выполнения индивидуального задания 2



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

"""
Создать класс Polinom для работы с многочленами до 100-й степени. Коэффициенты
должны быть представлены списком из 100 элементов-коэффициентов.
Младшая степень имеет меньший индекс (нулевая степень — нулевой индекс).
Размер списка задается как аргумент конструктора инициализации.
Реализовать арифметические операции и операции сравнения, вычисление значения
полинома для заданного значения, дифференцирование, интегрирование.
"""


6 usages
class Polynom:
    MAX_SIZE = 100
    def __init__(self, degree):
        # Степень полинома
        self.__degree = degree
        # Размер списка
        self.__size = degree + 1
        # Список коэффициентов
        self.__coef_list = []

    # Возвращает установленную длину
    def get_size(self):
        return self.__size

    # Получить индекс коэффициента полинома
    3 usages
    def get_index(self):
        print(self.__coef_list[-1])
        return self.__coef_list

    # Вывести коэффициенты полинома
    4 usages
    def print_coef(self):
        print(self.__coef_list)

    # Считываем значение коэффициентов
    2 usages
    def read_coef(self, strr=None):
        # Если на вход функции строка -> преобразуем
        if isinstance(strr, str):
            coeffs = strr.split(", ")
            if len(coeffs) <= Polynom.MAX_SIZE:
                for i in coeffs:
                    self.__coef_list.append(float(i))
        # Если на вход ничего не подано -> вводится по значению
        else:
            for i in range(self.__size):
                if self.__size <=Polynom.MAX_SIZE:
                    coef = float(input("Введите коэффициенты полинома: "))
                    self.__coef_list.append(coef)

    # Вычисление значения полинома для заданного значения x
    def count_value(self, x):
        res = []
        for idx, elem in enumerate(self.__coef_list):
            res.append(elem * x**idx)
        return sum(res)

    # Сумма двух полиномов
    2 usages
    def sum_polynom(self, pol_deg, index_pol2):
        if self.__degree > pol_deg:
            res_idx = [0] * self.__size
            index = 0
            for idx, elem in enumerate(index_pol2):
                res_idx[idx] = elem + self.__coef_list[idx]
                index = idx
            for index in range(index + 1, self.__size):
                res_idx[index] = self.__coef_list[index]
            return res_idx
        else:
            res_idx = [0] * (pol_deg + 1)
            index = 0
            for idx, elem in enumerate(self.__coef_list):
                res_idx[idx] = elem + index_pol2[idx]
                index = idx
            for index in range(index + 1, pol_deg + 1):
                res_idx[index] = index_pol2[index]
            return res_idx

    # Разность двух полиномов
    1 usage
    def sub_polynom(self, pol_deg, index_pol2):
        for idx in range(len(index_pol2)):
            index_pol2[idx] *= -1
        res = self.sum_polynom(pol_deg, index_pol2)
        return res

    # Произведение двух полиномов
    1 usage
    def multiply_pol(self, pol_deg, index_pol2):
        res_idx = [0] * (self.__degree + pol_deg + 1)
        for idx1, elem1 in enumerate(self.__coef_list):
            for idx2, elem2 in enumerate(index_pol2):
                res_idx[idx1 + idx2] += elem1 * elem2
        return res_idx

    # Умножение полинома на число (всех его коэффициентов)
    def multiply_pol_num(self, num):
        for idx in range(len(self.__coef_list)):
            self.__coef_list[idx] *= num
        return self.__coef_list
```

Рисунок 10 – Код программы индивидуального задания 2 (1)

```python
    # Проверка. Полиномы равны.
    def __eq__(self, val_pol2):
        if isinstance(val_pol2, Polynom):
            if self.__coef_list == val_pol2.__coef_list:
                return True
            else:
                return False

    # Проверка. Полином больше чем заданный
    def __gt__(self, val_pol2):
        if isinstance(val_pol2, Polynom):
            if self.__coef_list > val_pol2.__coef_list:
                return True
            else:
                return False

    # Проверка. Полином меньше чем заданный
    def __lt__(self, val_pol2):
        if self.__coef_list <= val_pol2.__coef_list:
            return True
        else:
            return False

    # Проверка. Полином не равен другому.
    def __ne__(self, val_pol2):
        if not self.__coef_list == val_pol2.__coef_list:
            return True
        else:
            return False

    # Дифференцирование полинома
    1 usage
    def differentiation(self):
        res = []
        for idx, elem in enumerate(self.__coef_list):
            if idx > 0:
                res.append(elem * idx)
        return res

    # Интегрирование полинома
    1 usage
    def integrate(self):
        res = [0] * (self.__size + 1)
        for idx, elem in enumerate(self.__coef_list):
            res[idx + 1] = elem / (idx + 1)
        return res

    # Получить конкретный коэффициент полинома
    def __getitem__(self, item):
        return self.__coef_list[item]
```

```python
if __name__ == "__main__":
    pol1 = Polynom(3)
    pol1.print_coef()
    pol1.read_coef()
    pol1.print_coef()
    pol2 = Polynom(3)
    pol2.print_coef()
    pol2.read_coef("1, 3, 4, 4")
    pol2.print_coef()

    print(f"Сумма: {pol1.sum_polynom(3, pol2.get_index())}")
    print(f"Разность: {pol1.sub_polynom(3, pol2.get_index())}")
    print(f"Произведение: {pol1.multiply_pol(3, pol2.get_index())}")
    # print(f"Полином 1, умноженный на 2: {pol1.multiply_pol_num(2)}")
    print(f"pol1 == pol2: {pol1 == pol2}")
    print(f"pol1 > pol2: {pol1 > pol2}")
    print(f"pol1 < pol2: {pol1 < pol2}")
    print(f"pol1 != pol2: {pol1 != pol2}")
    print(f"Дифференцирование полинома 1: {pol1.differentiation()}")
    print(f"Интегрирование полинома 1: {pol1.integrate()}")
```

Рисунок 11 – Код программы индивидуального задания 2 (1)

Вывод: Были приобретены и применены навыки по перегрузке операторов при написании программ с помощью языка программирования Python.

Контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

Для перегрузки операций в Python используются специальные методы, которые начинаются и заканчиваются двойным подчеркиванием. Например,

для перегрузки оператора сложения используется метод add, для оператора равенства - метод eq и т.д.

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