

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»  
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет о лабораторной работе №15 по дисциплине основы программной  
инженерии**

Выполнил:  
Выходцев Егор Дмитриевич,  
2 курс, группа ПИЖ-б-о-20-1,

Проверил:  
Доцент кафедры инфокоммуникаций,  
Воронкин Р.А.

Ставрополь, 2022 г

## 1. Примеры из методических указаний

```
ex1.py x e1.py x
7
8 class Vector2D:
9     def __init__(self, x, y):
10         self.x = x
11         self.y = y
12
13     def __repr__(self):
14         return 'Vector2D({}, {})'.format(self.x, self.y)
15
16     def __str__(self):
17         return '({}, {})'.format(self.x, self.y)
18
19     def __add__(self, other):
20         return Vector2D(self.x + other.x, self.y + other.y)
21
22     def __iadd__(self, other):
23         self.x += other.x
24         self.y += other.y
25         return self
26
27     def __sub__(self, other):
28         return Vector2D(self.x - other.x, self.y - other.y)
29
30     def __isub__(self, other):
31         self.x -= other.x
32         self.y -= other.y
33         return self
34
35     def __abs__(self):
36         return math.hypot(self.x, self.y)
37
38     def __bool__(self):
39         return self.x != 0 or self.y != 0
40
41     def __neg__(self):
42         return Vector2D(-self.x, -self.y)
43
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)
60
```

```
e1 x
C:\Users\student-09-525\PycharmProjec
(3, 4)
5.0
(5, 6)
(8, 10)
(-2, -2)
(-3, -4)
(8, 10)
True
False
(0, 0)

Process finished with exit code 0
```

```

e1.py x ex1.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
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                 elif a >= b:
24                     return gcd(a % b, b)
25                 else:
26                     return gcd(a, b % a)
27
28             sign = 1
29             if (self.__numerator > 0 > self.__denominator) or \
30                 (self.__numerator < 0 < self.__denominator):
31                 sign = -1
32
33             a, b = abs(self.__numerator), abs(self.__denominator)
34             c = gcd(a, b)
35             self.__numerator = sign * (a // c)
36             self.__denominator = b // c
37
38         # Клонировать дробь.
39         def __clone(self):
40             return Rational(self.__numerator, self.__denominator)
41
42         @property
43         def numerator(self):
44             return self.__numerator
45
46         @numerator.setter
47         def numerator(self, value):
48             self.__numerator = int(value)
49             self.__reduce()
50
51         @property
52         def denominator(self):
53             return self.__denominator
54
55         @denominator.setter

```

Rational > \_\_eq\_\_0

```

e1.py x ex1.py x
55 @denominator.setter
56 def denominator(self, value):
57     value = int(value)
58     if value == 0:
59         raise ValueError("Illegal value of the denominator")
60     self.__denominator = value
61     self.__reduce()
62
63     # Привести дробь к строке.
64 def __str__(self):
65     return f"{self.__numerator} / {self.__denominator}"
66
67 def __repr__(self):
68     return self.__str__()
69
70     # Привести дробь к вещественному значению.
71 def __float__(self):
72     return self.__numerator / self.__denominator
73
74     # Привести дробь к логическому значению.
75 def __bool__(self):
76     return self.__numerator != 0
77
78     # Сложение обыкновенных дробей.
79 def __iadd__(self, rhs): # +=
80     if isinstance(rhs, Rational):
81         a = self.numerator * rhs.denominator + \
82             self.denominator * rhs.numerator
83         b = self.denominator * rhs.denominator
84         self.__numerator, self.__denominator = a, b
85         self.__reduce()
86         return self
87     else:
88         raise ValueError("Illegal type of the argument")
89
90 def __add__(self, rhs): # +
91     return self.__clone().__iadd__(rhs)
92
93     # Вычитание обыкновенных дробей.
94 def __isub__(self, rhs): # -=
95     if isinstance(rhs, Rational):
96         a = self.numerator * rhs.denominator - \
97             self.denominator * rhs.numerator
98         b = self.denominator * rhs.denominator
99         self.__numerator, self.__denominator = a, b
100
101         self.__reduce()
102         return self
103     else:
104         raise ValueError("Illegal type of the argument")
105
106 def __sub__(self, rhs): # -
107     return self.__clone().__isub__(rhs)
108
109     # Умножение обыкновенных дробей.

```

Rational > \_\_eq\_\_()

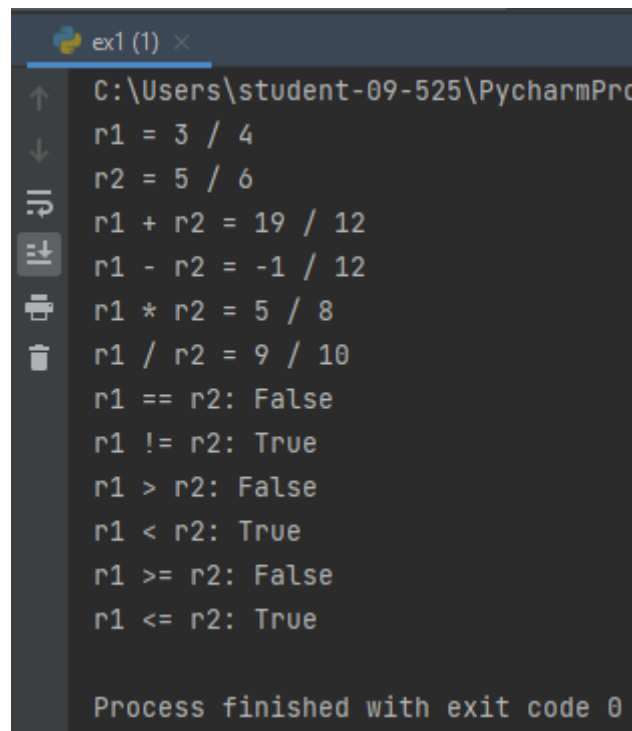
```

e1.py x ex1.py x
109 # Умножение обыкновенных дробей.
110 def __imul__(self, rhs): # *=
111     if isinstance(rhs, Rational):
112         a = self.numerator * rhs.numerator
113         b = self.denominator * rhs.denominator
114         self.__numerator, self.__denominator = a, b
115         self.__reduce()
116         return self
117     else:
118         raise ValueError("Illegal type of the argument")
119
120 def __mul__(self, rhs): # *
121     return self.__clone().__imul__(rhs)
122
123 # Деление обыкновенных дробей.
124 def __itruediv__(self, rhs): # /=
125     if isinstance(rhs, Rational):
126         a = self.numerator * rhs.denominator
127         b = self.denominator * rhs.numerator
128         if b == 0:
129             raise ValueError("Illegal value of the denominator")
130         self.__numerator, self.__denominator = a, b
131         self.__reduce()
132         return self
133     else:
134         raise ValueError("Illegal type of the argument")
135
136 def __truediv__(self, rhs): # /
137     return self.__clone().__itruediv__(rhs)
138
139 # Отношение обыкновенных дробей.
140 def __eq__(self, rhs): # ==
141     if isinstance(rhs, Rational):
142         return (self.numerator == rhs.numerator) and \
143             (self.denominator == rhs.denominator)
144     else:
145         return False
146
147 def __ne__(self, rhs): # !=
148     if isinstance(rhs, Rational):
149         return not self.__eq__(rhs)
150     else:
151         return False
152
153 def __gt__(self, rhs): # >
154     if isinstance(rhs, Rational):
155         return self.__float__() > rhs.__float__()
156     else:
157         return False
158
159 def __lt__(self, rhs): # <
160     if isinstance(rhs, Rational):
161         return self.__float__() < rhs.__float__()
162     else:
163         return False

```

Rational

```
163         return False
164
165     def __ge__(self, rhs): # >=
166         if isinstance(rhs, Rational):
167             return not self.__lt__(rhs)
168         else:
169             return False
170
171     def __le__(self, rhs): # <=
172         if isinstance(rhs, Rational):
173             return not self.__gt__(rhs)
174         else:
175             return False
176
177
178 ► if __name__ == '__main__':
179     r1 = Rational(3, 4)
180     print(f"r1 = {r1}")
181     r2 = Rational(5, 6)
182     print(f"r2 = {r2}")
183     print(f"r1 + r2 = {r1 + r2}")
184     print(f"r1 - r2 = {r1 - r2}")
185     print(f"r1 * r2 = {r1 * r2}")
186     print(f"r1 / r2 = {r1 / r2}")
187     print(f"r1 == r2: {r1 == r2}")
188     print(f"r1 != r2: {r1 != r2}")
189     print(f"r1 > r2: {r1 > r2}")
190     print(f"r1 < r2: {r1 < r2}")
191     print(f"r1 >= r2: {r1 >= r2}")
192     print(f"r1 <= r2: {r1 <= r2}")
193
```



```
ex1 (1) x
C:\Users\student-09-525\PycharmPro
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
```

## 2. Индивидуальное задание №1.

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

Перегруженные операторы:

`__init__(self, ...)`

`__lt__(self, other)` -  $x < y$  вызывает `x.__lt__(y)`.

`__le__(self, other)` -  $x \leq y$  вызывает `x.__le__(y)`.

`__eq__(self, other)` -  $x == y$  вызывает `x.__eq__(y)`.

`__ne__(self, other)` -  $x != y$  вызывает `x.__ne__(y)`

`__gt__(self, other)` -  $x > y$  вызывает `x.__gt__(y)`.

`__ge__(self, other)` -  $x \geq y$  вызывает `x.__ge__(y)`.

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

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

`__float__(self)` - приведение к float.

`__bool__(self)`



`__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 ( $x / y$ ).

`__iadd__(self, other)` -  $+=$ .

`__isub__(self, other)` -  $-=$ .

`__imul__(self, other)` -  $*=$ .

`__itruediv__(self, other)` -  $/=$ .

```
ind1.py x
1 ▶ 1 #!/usr/bin/env python3
2 2 # -*- coding: utf-8 -*-
3
4
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     def __reduce(self):
16         def gcd(a, b):
17             if a == 0:
18                 return b
19             elif b == 0:
20                 return a
21             elif a >= b:
22                 return gcd(a % b, b)
23             else:
24                 return gcd(a, b % a)
25
26         sign = 1
27         if (self.__numerator > 0 > self.__denominator) or \
28             (self.__numerator < 0 < self.__denominator):
29             sign = -1
30
31         a, b = abs(self.__numerator), abs(self.__denominator)
32         c = gcd(a, b)
33         self.__numerator = sign * (a // c)
34         self.__denominator = b // c
35
36     def __clone(self):
37         return Rational(self.__numerator, self.__denominator)
38
39     @property
```

Рисунок 1 – Код программы

```
ind1.py x
40 def numerator(self):
41     return self.__numerator
42
43 @numerator.setter
44 def numerator(self, value):
45     self.__numerator = int(value)
46     self.__reduce()
47
48 @property
49 def denominator(self):
50     return self.__denominator
51
52 @denominator.setter
53 def denominator(self, value):
54     value = int(value)
55     if value == 0:
56         raise ValueError("Illegal value of the denominator")
57     self.__denominator = value
58     self.__reduce()
59
60 def __str__(self):
61     return f"{self.__numerator} / {self.__denominator}"
62
63 def __repr__(self):
64     return self.__str__()
65
66 def __float__(self):
67     return self.__numerator / self.__denominator
68
69 def __bool__(self):
70     return self.__numerator != 0
71
72 def __iadd__(self, rhs):
73     if isinstance(rhs, Rational):
74         a = self.numerator * rhs.denominator + \
75             self.denominator * rhs.numerator
76         b = self.denominator * rhs.denominator
77         self.__numerator, self.__denominator = a, b
78         self.__reduce()
```

Рисунок 2 – Код программы, продолжение

```
ind1.py x
79         return self
80     else:
81         raise ValueError("Illegal type of the argument")
82
83     def __add__(self, rhs):
84         return self.__clone().__iadd__(rhs)
85
86     def __isub__(self, rhs):
87         if isinstance(rhs, Rational):
88             a = self.numerator * rhs.denominator - \
89                 self.denominator * rhs.numerator
90             b = self.denominator * rhs.denominator
91             self.__numerator, self.__denominator = a, b
92
93             self.__reduce()
94             return self
95         else:
96             raise ValueError("Illegal type of the argument")
97
98     def __sub__(self, rhs):
99         return self.__clone().__isub__(rhs)
100
101     def __imul__(self, rhs):
102         if isinstance(rhs, Rational):
103             a = self.numerator * rhs.numerator
104             b = self.denominator * rhs.denominator
105             self.__numerator, self.__denominator = a, b
106             self.__reduce()
107             return self
108         else:
109             raise ValueError("Illegal type of the argument")
110
111     def __mul__(self, rhs):
112         return self.__clone().__imul__(rhs)
113
114     def __itruediv__(self, rhs):
115         if isinstance(rhs, Rational):
116             a = self.numerator * rhs.denominator
117             b = self.denominator * rhs.numerator
```

Рисунок 3 – Код программы, продолжение

```
ind1.py x
118         if b == 0:
119             raise ValueError("Illegal value of the denominator")
120         self.__numerator, self.__denominator = a, b
121         self.__reduce()
122         return self
123     else:
124         raise ValueError("Illegal type of the argument")
125
126     def __truediv__(self, rhs):
127         return self.__clone().__itruediv__(rhs)
128
129     def __eq__(self, rhs):
130         if isinstance(rhs, Rational):
131             return (self.numerator == rhs.numerator) and \
132                 (self.denominator == rhs.denominator)
133         else:
134             return False
135
136     def __ne__(self, rhs):
137         if isinstance(rhs, Rational):
138             return not self.__eq__(rhs)
139         else:
140             return False
141
142     def __gt__(self, rhs):
143         if isinstance(rhs, Rational):
144             return self.__float__() > rhs.__float__()
145         else:
146             return False
147
148     def __lt__(self, rhs):
149         if isinstance(rhs, Rational):
150             return self.__float__() < rhs.__float__()
151         else:
152             return False
153
154     def __ge__(self, rhs):
155         if isinstance(rhs, Rational):
156             return not self.__lt__(rhs)
```

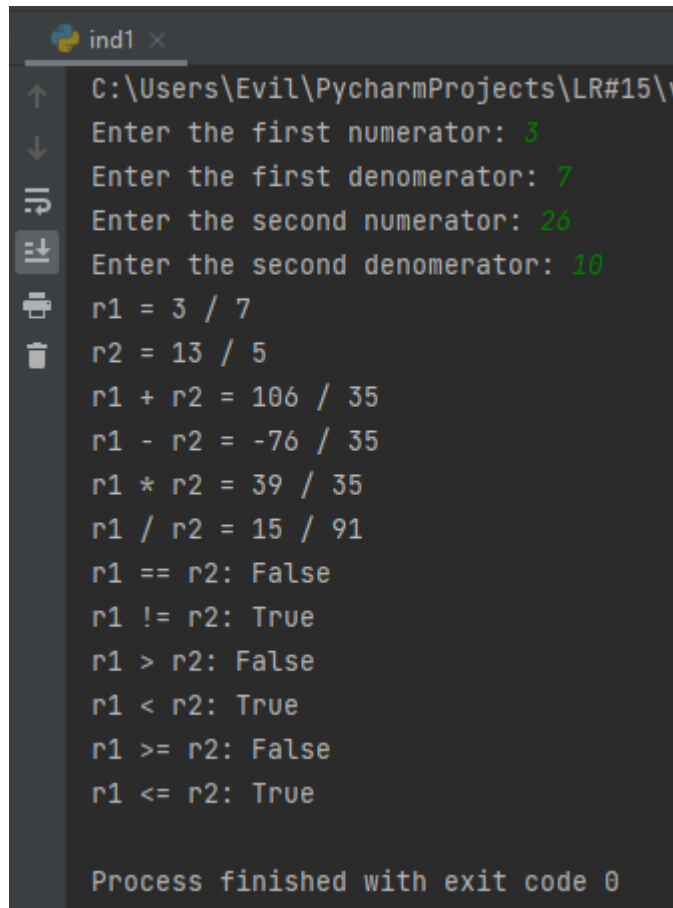
Рисунок 4 – Код программы, продолжение

```

156         return not self.__lt__(rhs)
157     else:
158         return False
159
160     def __le__(self, rhs):
161         if isinstance(rhs, Rational):
162             return not self.__gt__(rhs)
163         else:
164             return False
165
166
167 ► if __name__ == '__main__':
168     num1 = int(input("Enter the first numerator: "))
169     den1 = int(input("Enter the first denominator: "))
170     num2 = int(input("Enter the second numerator: "))
171     den2 = int(input("Enter the second denominator: "))
172     r1 = Rational(num1, den1)
173     print(f"r1 = {r1}")
174     r2 = Rational(num2, den2)
175     print(f"r2 = {r2}")
176     print(f"r1 + r2 = {r1 + r2}")
177     print(f"r1 - r2 = {r1 - r2}")
178     print(f"r1 * r2 = {r1 * r2}")
179     print(f"r1 / r2 = {r1 / r2}")
180     print(f"r1 == r2: {r1 == r2}")
181     print(f"r1 != r2: {r1 != r2}")
182     print(f"r1 > r2: {r1 > r2}")
183     print(f"r1 < r2: {r1 < r2}")
184     print(f"r1 >= r2: {r1 >= r2}")
185     print(f"r1 <= r2: {r1 <= r2}")
186

```

Рисунок 5 – Код программы, продолжение



```
ind1 x
C:\Users\Evil\PycharmProjects\LR#15\
Enter the first numerator: 3
Enter the first denominator: 7
Enter the second numerator: 20
Enter the second denominator: 10
r1 = 3 / 7
r2 = 13 / 5
r1 + r2 = 106 / 35
r1 - r2 = -76 / 35
r1 * r2 = 39 / 35
r1 / r2 = 15 / 91
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 6 – Результат выполнения программы

### 3. Индивидуальное задание №2. Вариант 5.

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.

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.

```
ind2.py x
1  ▶ 1 #!/usr/bin/env python3
2    2 # -*- coding: utf-8 -*-
3
4
5    class Polynom:
6        def __init__(self, degree):
7            self.__degree = degree
8            self.__size = degree + 1
9            self.__idx_list = []
10
11       def get_size(self):
12           return self.__size
13
14       def get_idx_list(self):
15           print(self.__idx_list[-1])
16           return self.__idx_list
17
18       def print_coef(self):
19           print(self.__idx_list)
20
21       def read_coef(self):
22           for i in range(self.__size):
23               c = float(input("Enter the coef: "))
24               self.__idx_list.append(c)
25
26       def count_x(self, x):
27           res = sum(elem * x ** idx for idx, elem in enumerate(self.__idx_list))
28           return res
29
30       def sum_polynom(self, pol_deg, idx_list2):
31           if self.__degree > pol_deg:
32               res_idx = [0] * self.__size
33               index = 0
34               for idx, elem in enumerate(idx_list2):
35                   res_idx[idx] = elem + self.__idx_list[idx]
36                   index = idx
37               for index in range(index + 1, self.__size):
38                   res_idx[index] = self.__idx_list[index]
39               return res_idx
40           else:
41               res_idx = [0] * (pol_deg + 1)
42               for i in range(pol_deg + 1):
43                   res_idx[i] = self.__idx_list[i]
44
45   if __name__ == '__main__':
46       pol_deg = int(input("Enter the degree: "))
47       pol = Polynom(pol_deg)
48       pol.read_coef()
49       pol.print_coef()
50       x = float(input("Enter the x: "))
51       res = pol.count_x(x)
52       print(res)
```

Рисунок 7 – Код программы



```

41         res_idx = [0] * (pol_deg + 1)
42         index = 0
43         for idx, elem in enumerate(self.__idx_list):
44             res_idx[idx] = elem + idx_list2[idx]
45             index = idx
46         for index in range(index + 1, pol_deg + 1):
47             res_idx[index] = idx_list2[index]
48         return res_idx
49
50     def pol_subtraction(self, pol_deg, idx_list2):
51         for idx in range(len(idx_list2)):
52             idx_list2[idx] *= -1
53         # print(idx_list2)
54         res = self.sum_polynom(pol_deg, idx_list2)
55         return res
56
57     def multiply_pol(self, pol_deg, idx_list2):
58         res_idx = [0] * (self.__degree + pol_deg + 1)
59         for idx1, elem1 in enumerate(self.__idx_list):
60             for idx2, elem2 in enumerate(idx_list2):
61                 res_idx[idx1 + idx2] += elem1 * elem2
62         return res_idx
63
64     def multiply_by_num(self, num):
65         for idx in range(len(self.__idx_list)):
66             self.__idx_list[idx] *= num
67         return self.__idx_list
68
69     def __eq__(self, val):
70         if isinstance(val, Polynom):
71             return self.__idx_list == val.__idx_list
72         else:
73             return len(self.__idx_list) == 1 and self.__idx_list[0] == val
74
75     def __gt__(self, val):
76         if isinstance(val, Polynom):
77             return self.__idx_list[-1] > val.__idx_list[-1]
78         else:
79             return len(self.__idx_list) == 1 and self.__idx_list[0] > val

```

Рисунок 8 – Код программы, продолжение

```
ind2.py x
79         return len(self.__idx_list) == 1 and self.__idx_list[0] > val
80
81     def __lt__(self, val):
82         if isinstance(val, Polynom):
83             if len(self.__idx_list) <= len(val.__idx_list):
84                 return self.__idx_list[-1] < val.__idx_list[-1]
85             else:
86                 return len(self.__idx_list) == 1 and self.__idx_list[0] > val
87
88     def __ne__(self, val):
89         if isinstance(val, Polynom):
90             return not self.__idx_list == val.__idx_list
91         else:
92             return not len(self.__idx_list) == 1 and self.__idx_list[0] == val
93
94     def differentiation(self):
95         res = []
96         for idx, elem in enumerate(self.__idx_list):
97             if idx > 0:
98                 res.append(elem * idx)
99         return res
100
101     def integrate(self):
102         res = [0] * (self.__size + 1)
103         for idx, elem in enumerate(self.__idx_list):
104             res[idx + 1] = elem / (idx + 1)
105         return res
106
107     def __getitem__(self, item):
108         return self.__idx_list[item]
109
110
111 if __name__ == '__main__':
112     pol = Polynom(4)
113     pol.print_coef()
114     pol.read_coef()
115     pol.print_coef()
116     print(f"Index overload: {pol[0]}, {pol[-1]}")
117     print(f"Polynom value: {pol.count_x(3)}")
118     pol2 = Polynom(5)
119     pol2.read_coef()
120
121 if __name__ == '__main__':
```

Рисунок 9 – Код программы, продолжение

```

119     pol2.read_coef()
120     pol2.print_coef()
121     print(f"Sum of polynoms: {pol.sum_polynom(5, pol2.get_idx_list())}")
122     print(f"Polynom subtract: {pol.pol_subtraction(5, pol2.get_idx_list())}")
123     print(
124         f"Polynom multiplication: {pol.multiply_pol(5, pol2.get_idx_list())}"
125     )
126     print(f"Polynom mult. by number: {pol.multiply_by_num(5)}")
127     pol3 = Polynom(4)
128     pol3.read_coef()
129     pol3.print_coef()
130     print(f"pol == pol3: {pol == pol3}")
131     print(f"pol > pol3: {pol > pol3}")
132     print(f"pol < pol3: {pol < pol3}")
133     print(f"pol != pol3: {pol != pol3}")
134     print(f"Polynom differentiation: {pol.differentiation()}")
135     print(f"Polynom integration: {pol.integrate()}")
136

```

Рисунок 10 – Код программы, продолжение

```
ind2 x
C:\Users\Evil\PycharmProjects\LR#15\venv\Scripts\python.exe C:/Users/Evil/PycharmProjects/LR#
[]
Enter the coef: 2
Enter the coef: 3
Enter the coef: 4
Enter the coef: 5
Enter the coef: 6
[2.0, 3.0, 4.0, 5.0, 6.0]
Index overload: 2.0, 6.0
Polynom value: 668.0
Enter the coef: 1
Enter the coef: 2
Enter the coef: 3
Enter the coef: 4
Enter the coef: 5
Enter the coef: 6
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
6.0
Sum of polynoms: [3.0, 5.0, 7.0, 9.0, 11.0, 6.0]
6.0
Polynom subtract: [1.0, 1.0, 1.0, 1.0, 1.0, -6.0]
-6.0
Polynom multiplication: [-2.0, -7.0, -16.0, -30.0, -50.0, -70.0, -76.0, -73.0, -60.0, -36.0]
Polynom mult. by number: [10.0, 15.0, 20.0, 25.0, 30.0]
Enter the coef: 23
Enter the coef: 67
Enter the coef: 335
Enter the coef: 235
Enter the coef: 2357
[23.0, 67.0, 335.0, 235.0, 2357.0]
pol == pol3: False
pol > pol3: False
pol < pol3: True
pol != pol3: True
Polynom differentiation: [15.0, 40.0, 75.0, 120.0]
Polynom integration: [0, 10.0, 7.5, 6.666666666666667, 6.25, 6.0]

Process finished with exit code 0
```

Рисунок 11 – Результат выполнения программы

#### 4. Ответы на контрольные вопросы

1. Какие средства существуют в Python для перегрузки операций?

ЗаклЮчение опрeатора 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, использующиеся для внутреннего представления в python.