

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

ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчёт по лабораторной работе №4.2 по дисциплине:

Основы программной инженерии

Выполнила:

студент группы ПИЖ-б-о-20-1

Лазарева Дарья Олеговна

Проверил:

доцент кафедры инфокоммуникаций

Романкин Р.А.

Ставрополь, 2022 г.

ВЫПОЛНЕНИЕ:

1. Примеры из методических указаний

```

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
26 if __name__ == '__main__':
27     v1 = Vector2D(3, 4)
28     v2 = Vector2D(5, 6)
29     v3 = Vector2D(8, 10)
30     v4 = Vector2D(-2, -2)
31     v5 = Vector2D(-3, -4)
32     v6 = Vector2D(8, 10)
33     v7 = Vector2D(0, 0)
34     print(v1 + v2)
35     print(v3 + v4)
36     print(v5 + v6)
37     print(v7 + v8)
38     print(v9 + v10)
39     print(v11 + v12)
40     print(v13 + v14)
41     print(v15 + v16)
42     print(v17 + v18)
43     print(v19 + v20)
44     print(v21 + v22)
45     print(v23 + v24)
46     print(v25 + v26)
47     print(v27 + v28)
48     print(v29 + v30)
49     print(v31 + v32)
50     print(v33 + v34)
51     print(v35 + v36)
52     print(v37 + v38)
53     print(v39 + v40)
54     print(v41 + v42)
55     print(v43 + v44)
56     print(v45 + v46)
57     print(v47 + v48)
58     print(v49 + v50)
59     print(v51 + v52)
60     print(v53 + v54)
61     print(v55 + v56)
62     print(v57 + v58)
63     print(v59 + v60)
64     print(v61 + v62)
65     print(v63 + v64)
66     print(v65 + v66)
67     print(v67 + v68)
68     print(v69 + v70)
69     print(v71 + v72)
70     print(v73 + v74)
71     print(v75 + v76)
72     print(v77 + v78)
73     print(v79 + v80)
74     print(v81 + v82)
75     print(v83 + v84)
76     print(v85 + v86)
77     print(v87 + v88)
78     print(v89 + v90)
79     print(v91 + v92)
80     print(v93 + v94)
81     print(v95 + v96)
82     print(v97 + v98)
83     print(v99 + v100)
84     print(v101 + v102)
85     print(v103 + v104)
86     print(v105 + v106)
87     print(v107 + v108)
88     print(v109 + v110)
89     print(v111 + v112)
90     print(v113 + v114)
91     print(v115 + v116)
92     print(v117 + v118)
93     print(v119 + v120)
94     print(v121 + v122)
95     print(v123 + v124)
96     print(v125 + v126)
97     print(v127 + v128)
98     print(v129 + v130)
99     print(v131 + v132)
100    print(v133 + v134)
101    print(v135 + v136)
102    print(v137 + v138)
103    print(v139 + v140)
104    print(v141 + v142)
105    print(v143 + v144)
106    print(v145 + v146)
107    print(v147 + v148)
108    print(v149 + v150)
109    print(v151 + v152)
110    print(v153 + v154)
111    print(v155 + v156)
112    print(v157 + v158)
113    print(v159 + v160)
114    print(v161 + v162)
115    print(v163 + v164)
116    print(v165 + v166)
117    print(v167 + v168)
118    print(v169 + v170)
119    print(v171 + v172)
120    print(v173 + v174)
121    print(v175 + v176)
122    print(v177 + v178)
123    print(v179 + v180)
124    print(v181 + v182)
125    print(v183 + v184)
126    print(v185 + v186)
127    print(v187 + v188)
128    print(v189 + v190)
129    print(v191 + v192)
130    print(v193 + v194)
131    print(v195 + v196)
132    print(v197 + v198)
133    print(v199 + v200)
134    print(v201 + v202)
135    print(v203 + v204)
136    print(v205 + v206)
137    print(v207 + v208)
138    print(v209 + v210)
139    print(v211 + v212)
140    print(v213 + v214)
141    print(v215 + v216)
142    print(v217 + v218)
143    print(v219 + v220)
144    print(v221 + v222)
145    print(v223 + v224)
146    print(v225 + v226)
147    print(v227 + v228)
148    print(v229 + v230)
149    print(v231 + v232)
150    print(v233 + v234)
151    print(v235 + v236)
152    print(v237 + v238)
153    print(v239 + v240)
154    print(v241 + v242)
155    print(v243 + v244)
156    print(v245 + v246)
157    print(v247 + v248)
158    print(v249 + v250)
159    print(v251 + v252)
160    print(v253 + v254)
161    print(v255 + v256)
162    print(v257 + v258)
163    print(v259 + v260)
164    print(v261 + v262)
165    print(v263 + v264)
166    print(v265 + v266)
167    print(v267 + v268)
168    print(v269 + v270)
169    print(v271 + v272)
170    print(v273 + v274)
171    print(v275 + v276)
172    print(v277 + v278)
173    print(v279 + v280)
174    print(v281 + v282)
175    print(v283 + v284)
176    print(v285 + v286)
177    print(v287 + v288)
178    print(v289 + v290)
179    print(v291 + v292)
180    print(v293 + v294)
181    print(v295 + v296)
182    print(v297 + v298)
183    print(v299 + v300)
184    print(v301 + v302)
185    print(v303 + v304)
186    print(v305 + v306)
187    print(v307 + v308)
188    print(v309 + v310)
189    print(v311 + v312)
190    print(v313 + v314)
191    print(v315 + v316)
192    print(v317 + v318)
193    print(v319 + v320)
194    print(v321 + v322)
195    print(v323 + v324)
196    print(v325 + v326)
197    print(v327 + v328)
198    print(v329 + v330)
199    print(v331 + v332)
200    print(v333 + v334)
201    print(v335 + v336)
202    print(v337 + v338)
203    print(v339 + v340)
204    print(v341 + v342)
205    print(v343 + v344)
206    print(v345 + v346)
207    print(v347 + v348)
208    print(v349 + v350)
209    print(v351 + v352)
210    print(v353 + v354)
211    print(v355 + v356)
212    print(v357 + v358)
213    print(v359 + v360)
214    print(v361 + v362)
215    print(v363 + v364)
216    print(v365 + v366)
217    print(v367 + v368)
218    print(v369 + v370)
219    print(v371 + v372)
220    print(v373 + v374)
221    print(v375 + v376)
222    print(v377 + v378)
223    print(v379 + v380)
224    print(v381 + v382)
225    print(v383 + v384)
226    print(v385 + v386)
227    print(v387 + v388)
228    print(v389 + v390)
229    print(v391 + v392)
230    print(v393 + v394)
231    print(v395 + v396)
232    print(v397 + v398)
233    print(v399 + v400)
234    print(v401 + v402)
235    print(v403 + v404)
236    print(v405 + v406)
237    print(v407 + v408)
238    print(v409 + v410)
239    print(v411 + v412)
240    print(v413 + v414)
241    print(v415 + v416)
242    print(v417 + v418)
243    print(v419 + v420)
244    print(v421 + v422)
245    print(v423 + v424)
246    print(v425 + v426)
247    print(v427 + v428)
248    print(v429 + v430)
249    print(v431 + v432)
250    print(v433 + v434)
251    print(v435 + v436)
252    print(v437 + v438)
253    print(v439 + v440)
254    print(v441 + v442)
255    print(v443 + v444)
256    print(v445 + v446)
257    print(v447 + v448)
258    print(v449 + v450)
259    print(v451 + v452)
260    print(v453 + v454)
261    print(v455 + v456)
262    print(v457 + v458)
263    print(v459 + v460)
264    print(v461 + v462)
265    print(v463 + v464)
266    print(v465 + v466)
267    print(v467 + v468)
268    print(v469 + v470)
269    print(v471 + v472)
270    print(v473 + v474)
271    print(v475 + v476)
272    print(v477 + v478)
273    print(v479 + v480)
274    print(v481 + v482)
275    print(v483 + v484)
276    print(v485 + v486)
277    print(v487 + v488)
278    print(v489 + v490)
279    print(v491 + v492)
280    print(v493 + v494)
281    print(v495 + v496)
282
```

```
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             if b == 0:
22                 return a
23             return gcd(b, a % b)
24
25         g = gcd(self.__numerator, self.__denominator)
26         self.__numerator //= g
27         self.__denominator //= g
28
29 if __name__ == '__main__':
30     r1 = Rational(3, 4)
31     r2 = Rational(5, 6)
32     print(r1 + r2)
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)
```

ex2 x

C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/4.2/ex2.py

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

Индивидуальное задание №1

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

```
#!/usr/bin/env python 3
# -*- coding: utf-8 -*-

class Num:

    def __init__(self, first, second):
        self.first = first
        self.second = second
```

```

        if self.first == 0:
            raise ValueError

    def __pow__(self, other):
        a = self.first + self.second
        b = other.first + other.second
        return a ** b

if __name__ == "__main__":
    num1 = Num(3.4, 0)
    num2 = Num(2.7, 0)
    print(f"Результат: {num1 ** num2}")

```

```

C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/4.2/ind1.py
Результат: 27.226579002152647

```

Индивидуальное 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.

Создать класс BitString для работы с битовыми строками не более чем из 100 бит. Битовая строка должна быть представлена списком типа int, каждый элемент которого принимает значение 0 или 1. Реальный размер списка задается как аргумент конструктора инициализации. Должны быть реализованы все традиционные операции для работы с битовыми строками: and, or, xor, not. Реализовать сдвиг влево и сдвиг вправо на заданное количество битов.

```

#!/usr/bin/env python 3
# -*- coding: utf-8 -*-

class BitString:

    #Инициализация
    def __init__(self, x):
        self.size = x
        self.x = [0] * self.size

    #Установка значения
    def set(self, x):
        self.x = list(map(int, f'{x:b}'.rjust(self.size, '0')))

```

```

#Оператор not (~)
def __invert__(self):
    self.x = [int(not i) for i in self.x]
    return self

#Оператор or (|)
def __or__(self, other):
    x = [a | b for a, b in zip(self.x, other.x)]
    return ''.join(map(str, x))

#Оператор xor (^)
def __xor__(self, other):
    x = [a ^ b for a, b in zip(self.x, other.x)]
    return ''.join(map(str, x))

#Оператор and (&)
def __and__(self, other):
    x = [a & b for a, b in zip(other.x, self.x)]
    return ''.join(map(str, x))

#Оператор сдвиг влево (<<)
def __lshift__(self, x):
    del (self.x[0:x])
    self.x += [0] * x
    return self

#Оператор сдвиг вправо (>>)
def __rshift__(self, x):
    del (self.x[len(self.x) - x:])
    self.x = [0] * x + self.x
    return self

#Вывод результата в консоль
def __str__(self):
    return ''.join(map(str, self.x))

if __name__ == "__main__":
    x = BitString(8) # Размер списка 1 - 8 бит
    y = BitString(8) # Размер списка 2 - 8 бит

    x.set(55) # Первая цифра 00110111
    print(x)
    y.set(27) # Вторая цифра 00011011
    print(y)

    print(f'{x} and {y} = {x & y}')
    print(f'{x} or {y} = {x | y}')
    print(f'{x} xor {y} = {x ^ y}')
    print(f'{x} not = {~x}')
    print(f'{y} >> 1 = {y >> 1}')
    print(f'{x} << 2 = {x << 2}')

```

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/4.2/ind2.py
00110111
00011011
00110111 and 00011011 = 00010011
00110111 or 00011011 = 00111111
00110111 xor 00011011 = 00101100
00110111 not = 11001000
00011011 >> 1 = 00001101
11001000 << 2 = 00100000
```

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?

__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 ($\text{divmod}(x, y)$).

__pow__(self, other[, modulo]) - возведение в степень ($x ** y$, $\text{pow}(x, y[, \text{modulo}])$).

__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.