Методы списков

Операция	Описание	Пример
`x in a`	Проверка, что x содержится в a	`5 in [2, 3, 5]`
`x not in a`	Проверка, что x не содержится в a То же, что и `not (x in a)`	`5 not in [2, 3, 6]`
`a + a2`	Конкатенация списков, то есть новый список, в котором сначала идут все элементы a, а затем все элементы a2	`[2, 4] + [5, 3] == [2, 4, 5, 3]`
`a * k`	Список a, повторенный k раз	`[2, 3] * 3 == [2, 3, 2, 3, 2, 3]`
`a[n]`	n-й элемент списка, отрицательные n — для отсчета с конца	`[2, 3, 7][0] == 2` `[2, 3, 7][-1] == 7`
`a[start:stop:step]`	Срез списка	`[2, 3, 7][:2] == [2, 3]`
`len(a)`	Длина списка	`len([2, 3, 7]) == 3`
`max(a)`	Максимальный элемент списка	`max([2, 3, 7]) == 7`
`min(a)`	Минимальный элемент списка	`min([2, 3, 7]) == 2`
`sum(a)`	Сумма элементов списка	`sum([2, 3, 7]) == 12`
`a.index(x)`	Индекс первого вхождения x в a (вызовет ошибку, если `x not in a`, то есть если x отсутствует в a)	`[2, 3, 7].index(7) == 2`
`a.count(x)`	Количество вхождений x в a	`[2, 7, 3, 7].count(7) == 2`

`a.append(x)`	Добавить x в конец a	`a = [2, 3, 7]`
		`a.append(8)`
		`a == [2, 3, 7, 8]`

`a.extend(a2)`	Добавить элементы коллекции a2 в конец a	`a = [2, 3, 7]`
		`a.extend([8, 4, 5])`
		`a == [2, 3, 7, 8, 4, 5]`

`del a[n]`	Удалить n-й элемент списка	`a = [2, 3, 7]`
		`del a[1]`
		`a == [2, 7]`

`del a[start:stop:step]`	Удалить из a все элементы, попавшие в срез	`a = [2, 3, 7]`
		`del a[:2]`
		`a == [7]`

`a.clear()`	Удалить из a все элементы (то же, что `del a[:]`)	`a.clear()`
`a.copy()`	Копия a (то же, что и полный срез `a[:]`)	`b = a.copy()`
`a += a2` `a *= k`	Заменить содержимое списка на `a + a2` и `a * k` соответственно	

`a.insert(n, x)`	Вставить x в a на позицию n, подвинув последующую часть дальше	`a = [2, 3, 7]`
		`a.insert(0, 8)`
		`a == [8, 2, 3, 7]`

`a.pop(n)`	Получить n-й элемент списка и одновременно удалить его из списка. Вызов метода без аргументов равносилен удалению последнего элемента:
	`a.pop() == a.pop(-1)`
		`a = [2, 3, 7]`
		`a.pop(1) == 3`
		`a == [2, 7]`

`a.remove(x)`	Удалить первое вхождение x в a, в случае `x not in a` — ошибка	`a = [2, 3, 7]`
		`a.remove(3)`
		`a == [2, 7]`

`a.reverse()`	Изменить порядок элементов в `a` на обратный (перевернуть список)	`a = [2, 3, 7]`
		`a.reverse()`
		`a == [7, 3, 2]`

`a.sort()`	Отсортировать список по возрастанию	`a = [3, 2, 7]`
		`a.sort()`
		`a == [2, 3, 7]`

`a.sort(reverse=True)`	Отсортировать список по убыванию	`a = [3, 2, 7]`
		`a.sort(reverse = True)`
		`a == [7, 3, 2]`

`bool(a)`	Один из способов проверить список на пустоту (возвращает `True`, если список непустой, и `False` в противном случае)	

Методы строк

Операция	**Описание**	**Пример**
`s2 in s`	Проверка, что подстрока `s2` содержится в `s`	`'m' in 'team'`
`s2 not in s`	Проверка, что подстрока `s2` не содержится в `s` то же, что `not (s2 in s)`	`'I' not in 'team'`
`s + s2`	Конкатенация (склейка) строк, то есть строка, в которой сначала идут все символы из `s`, а затем все символы из `s2`	`'tea' + 'm' == 'team'`
`s * k`	Строка `s`, повторенная `k` раз	`'ha' * 3 == 'hahaha'`
`s[n]`	`n`-й элемент строки, отрицательные `n` — для отсчета с конца	`'team'[2] == 'a'` `'team'[-1] == 'm'`
`s[start:stop:step]`	Срез строки	`'mama'[:2] == 'ma'`
`len(s)`	Длина строки	`len('abracadabra') == 11`

`s.find(s2)`
`s.rfind(s2)`
	Индекс начала первого или последнего
	вхождения
	подстроки `s2` в `s` (вернет -1, если `s2 not in s`)
		`s = 'abracadabra'`
		`s.find('ab') == 0`
		`s.rfind('ab') == 7`
		`s.find('x') == -1`

`s.count(s2)`	Количество неперекрывающихся вхождений `s2` в `s`	`'abracadabra'.count('a') == 5`
`s.startswith(s2)` `s.endswith(s2)`	Проверка, что `s` начинается с `s2` или оканчивается на `s2`	`'abracadabra'.startswith('abra')`
`s += s2` `s *= k`	Заменить содержимое строки на `s + s2` и `s * k` соответственно	

`s.isdigit()`
`s.isalpha()`
`s.isalnum()`
	Проверка, что в строке `s` все символы — цифры,
	буквы (включая кириллические),
	цифры или буквы
	соответственно
		`'100'.isdigit()`
		`'abc'.isalpha()`
		`'E315'.isalnum()`

`s.islower()` `s.isupper()`	Проверка, что в строке `s` не встречаются большие буквы, маленькие буквы. Обратите внимание, что для обеих этих функций знаки препинания и цифры дают True	`'hello!'.islower()` `'123PYTHON'.isupper()`
`s.lower()` `s.upper()`	Строка `s`, в которой все буквы (включая кириллические) приведены к верхнему или нижнему регистру, т. е. заменены на строчные (маленькие) или заглавные (большие)	`'Привет!'.lower() == 'привет!'` `'Привет!'.upper() == 'ПРИВЕТ!'`
`s.capitalize()`	Строка `s`, в которой первая буква — заглавная	`'привет'.capitalize() == 'Привет'`

`s.lstrip()`
`s.rstrip()`
`s.strip()`
	Строка `s`, у которой удалены символы пустого пространства (пробелы, табуляции) в начале,
	в конце или с обеих сторон
		`' Привет! '.strip() == 'Привет!'`

`s.ljust(k, c)` `s.rjust(k, c)`	Добавляет справа или слева нужное количество символов `c`, чтобы длина `s` достигла `k`	`'Привет'.ljust(8, '!') == 'Привет!!'`
`s.join(a)`	Склеивает строки из списка `a` через символ `s`	`'+'.join(['Вася', 'Маша']) == 'Вася+Маша'`
`s.split(s2)`	Список всех слов строки `s` (подстрок, разделенных строками `s2`)	`'Раз два три!'.split('а') == ['Р', 'з дв', ' три!']`
`s.replace(s2, s3)`	Строка `s`, в которой все неперекрывающиеся вхождения `s2` заменены на `s3` Есть необязательный третий параметр, с помощью которого можно указать, сколько раз производить замену	`'Раз два три!'.replace('а', 'я') =='Ряз двя три!'` `'Раз два три!'.replace('а', 'я', 1) == 'Ряз два три!'`
`list(s)`	Список символов из строки строки `s`	`list('Привет') == ['П', 'р', 'и', 'в', 'е', 'т']`
`bool(s)`	Проверка, что строка не пустая (возвращает `True`, если не пустая, и `False` в противном случае)	
`int(s)` `float(s)`	Если в строке `s` записано целое (дробное) число, получить это число, иначе — ошибка	`int('25') == 25`
`str(x)`	Представить любой объект `x` в виде строки	`str(25) == '25'`