Урок №12. Методы списков и строк

План урока

1. Как пользоваться таблицами.
2. Основные методы списков.
3. Основные методы строк.
4. Стек.
5. Приведение к типу bool.
6. Цепочки вызовов методов.
7. Функции dir и help.

Аннотация ¶

В материалах урока приводятся таблицы с почти полным перечнем методов списков и строк, которые можно использовать как справочный материал. Рассматривается неявное приведение объектов к булеву типу. Приводятся примеры цепочек вызова методов. Появление метода `pop` позволяет познакомить учеников с понятием стека.

Этот урок отличается от прочих: большую часть его материала не нужно запоминать, но можно использовать как справочный материал. С этой же целью вводятся функции `dir` и `help`.

Как пользоваться таблицами

Мы уже знакомы с некоторыми функциями для работы со списками и строками. Также мы встречались с методом `append`, который позволяет добавлять элементы в список, методом `split` для разбиения строки на список «слов» и методом `join` для «сшивки» списка строк в одну. Однако мы не знаем, как делать со списками многие другие вещи (добавлять элементы в произвольное место, удалять элементы и т. д.). Такие задачи решаются с помощью других функций и методов.

Далее приводится перечень основных методов списков и строк, а также функций и специальных синтаксических конструкций для работы с ними, включая уже знакомые нам. Если при написании программы вы забудете, как называется тот или иной метод или какие у него аргументы, смело заглядывайте в материалы этого урока, в документацию (https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range), пользуйтесь поиском в интернете или описанными здесь функциями `dir` и `help`. Главное — усвоить основные возможности стандартных типов строки и списка, а детали (например, порядок аргументов в методе `insert` или название именованного аргумента в методе `split`) всегда можно посмотреть здесь или в документации.

В таблице предполагается, что `a` и `a2` — списки, `x` — элемент списка, `s` и `s2` — строки, `c` — символ (строка длины 1), n — индекс элемента списка или строки, start, stop, step — границы среза (т. е. тоже индексы) и шаг среза, k — натуральное число.

Также для методов и функций даны примеры выражений, которые будут истинными `True` и демонстрируют действие этого метода. Например, выражения `5 in [2, 3, 5]` и `'abc'.isalpha()` равны `True`. Их можно подставить в оператор `if` (и соответствующий блок будет выполнен) или в функцию `print` (и тогда она выведет `True`).

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, 5]`
`a + a2`	Конкатенация списков, то есть 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 отсчёта с конца	`[2, 3, 7][0] == 2` `[2, 3, 7][-1] == 7`
`a[start:stop:step]`	срез (см. урок №10)	`[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]`

`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 `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]`

Операция	Описание	Пример

`a.pop(n)`	получить n-й элемент списка и одновременно удалить его из списка
	`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]`

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	`'oo' * 3 = 'oooooo'`
`s[n]`	`n`-й элемент строки, отрицательные `n` — для отсчёта с конца	`'team'[2] == 'a'` `team[-1] == 'm'`
`s[start:stop:step]`	срез (см. урок №10)	`'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` (вернёт -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 `s` 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.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.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!!'`

Операция	Описание	Пример
`s.join(a)`	a — список строк, тогда `s.join(a)` — эти строки, «склеенные» через s	`'+'.join(['Вася', 'Маша']) == 'Вася+Маша'`
`s.split(s2)`	Список слов строки s (подстрок, разделённых 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	`'Раз два  три!'.replace('а', 'я') == 'Ряз двя  три!'`
`list(s)`	Список символов из строки строки s	`list('Привет') == ['П', 'р', 'и', 'в', 'е', 'т']`
`bool(s)`	Проверка, что строка не пустая	
`int(s)` `float(s)`	Если в строке s записано целое (дробное) число, получить это число, иначе ошибка	`int('25') == 25`
`str(x)`	Получить какое-то представление в виде строки любого объекта x	`str(25) == '25'`

Обратите внимание, что **никакие** методы строк, включая `s.replace(...)`, не изменяют саму строку `s`. Все они лишь возвращают **изменённую** строку, в отличие от большинства методов списков.

`a.sort()`, например, ничего не возвращает, а изменяет сам список a.

Стек

Приведем несколько примеров использования самых популярных методов.

Представьте себе стопку сложенных футболок или любых других не одинаковых предметов, которые можно сложить в стопку. Из стопки удобно забрать самый верхний предмет — тот, который положили в неё последним. Если вы полностью разберёте стопку, то будете брать футболки в порядке, обратном тому, в котором их в неё клали. Если пользоваться только методами `append` и `pop` без аргументов, то список оказывается как раз такой «стопкой».

Такая структура данных называется **стек** (`stack`, что и значит «стопка»). По-английски, такую организацию называют **LIFO** - Last In First Out - **Первый Пришел Последним Ушел!**

Так, в примере ниже порядок вывода будет обратным порядку ввода:

```
stack = []
for i in range(5):
    print('Какую футболку вы кладёте сверху стопки?')
    stack.append(input())
while stack:  # пока стопка не закончилась
    print('Сегодня вы надеваете футболку', stack.pop())
```

```
Какую футболку вы кладёте сверху стопки?
белую
Какую футболку вы кладёте сверху стопки?
красную
Какую футболку вы кладёте сверху стопки?
синюю
Какую футболку вы кладёте сверху стопки?
зеленую
Какую футболку вы кладёте сверху стопки?
веселую
Сегодня вы надеваете футболку веселую
Сегодня вы надеваете футболку зеленую
Сегодня вы надеваете футболку синюю
Сегодня вы надеваете футболку красную
Сегодня вы надеваете футболку белую
```

Здесь использовалась конструкция **while stack:**. В условии оператора `if` или `while` любой объект интерпретируется как `bool`: либо как `True`, либо как `False`. В случае списков и строк в `False` превращаются только [] и '', соответственно (а в случае чисел — только ноль).

Иными словами, можно было бы написать `while bool(stack):` или `while stack != []:` и получить тот же самый результат, но самый короткий и общепринятый вариант — просто `while stack:`.

Цепочки вызовов

Бывает удобно строить последовательные цепочки вызовов методов. Например, `s.strip().lower().replace('ё', 'е')` выдаст строку `s`, в которой убраны начальные и конечные пробелы, все буквы сделаны маленькими, после чего убраны все точки над «ё». В результате этого преобразования строка **' Зелёный клён '** превратится в **'зеленый клен'**.

```
'мало Средне    МНОГО'.lower().split().index('средне')
```

```
1
```

В данном примере строка сначала полностью приводится к нижнему регистру (все буквы — маленькие) при помощи метода `lower()`. Затем она превращается в список слов `['мало', 'средне', 'много']` при помощи `split()`. Далее метод `index` находит в этом списке слово «средне» на позиции номер 1.

Функции dir и help

Список всех методов любого объекта (в том числе тех, о которых вы вряд ли хотели узнать) можно получить с помощью функции **dir**. Например, `dir([])` вернёт все методы списков, и оператор `print(dir([]))` выведет длинный список, оканчивающийся так: `'index', 'insert', 'pop', 'remove', 'reverse', 'sort']`.

Функция **help** выводит на экран справку по любому типу или отдельному методу. `help([])` выведет на экран много информации, большая часть которой пока лишняя. А вот `help([].insert)` выведет на экран краткую справку именно по методу `insert`, подсказывая, что первый аргумент этого метода — индекс, а второй — тот объект, который нужно вставить в список на этот индекс. Заметьте, после `insert` нет скобок (...) — ведь мы не хотим вызвать этот метод, чтобы вставить что-то в какой-то список. Функции `help` в качестве аргумента передаётся сам метод `insert`.