
Яндекс

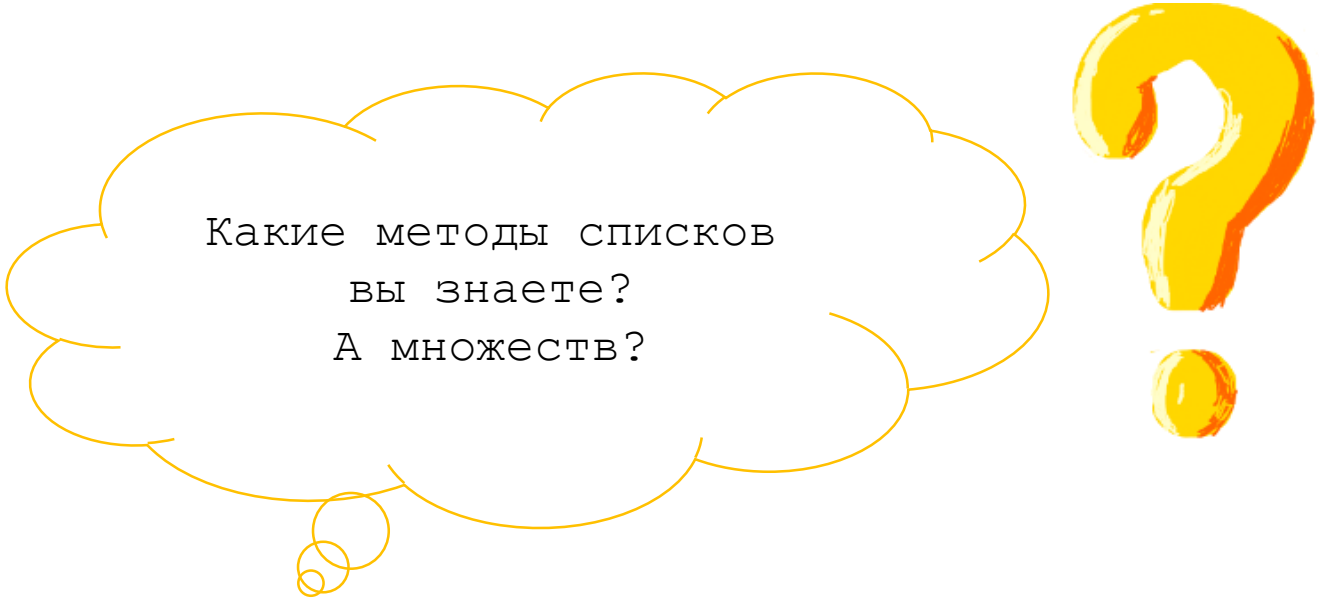
Академия Яндекса

Списочные выражения. Методы split и join

Методы split и join

Методы `split` и `join`

Изучая множества и списки, мы уже неоднократно встречались с методами – функциями, «приклеенными» к объекту (списку или множеству) и изменяющими его содержимое.



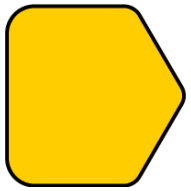
Методы `split` и `join`

Методы есть не только у списков и множеств, но и у строк. Сегодня мы изучим два очень полезных метода строк: `split` и `join`.

Они противоположны по смыслу: `split` разбивает строку по произвольному разделителю на список «слов», а `join` собирает из списка слов единую строку через заданный разделитель.

Синтаксис:

После имени переменной, содержащей объект-строку (или просто после строки), через точку пишется имя метода, затем в круглых скобках указываются аргументы.



`split` и `join`, в отличие, например, от метода списков `append` или метода множеств `add`, не изменяют объект, которому принадлежат, а создают **новый** (список или строку, соответственно) и возвращают его, как это делают обычные функции типа `len`.

Метод `split`

В этом примере все сравнения истинны, т. е. все вызовы функции `print` выведут `True`.

```
s = 'раз два три'
print(s.split() == ['раз', 'два', 'три'])
print('     one two  three  '.split() == ['one', 'two', 'three'])
print('192.168.1.1'.split('.') == ['192', '168', '1', '1'])
print(s.split('а') == ['р', 'з дв', ' три'])
print('A##B##C'.split('##') == ['A', 'B', 'C'])
```

Метод `split`

Метод **`split`** можно вызвать вообще без аргументов или с одним аргументом-строкой. В первом случае строка разбивается на части, разделённые любыми символами пустого пространства (набором пробелов, символом табуляции и т. д.). Во втором случае разделителем слов считается строка-аргумент. Из получившихся слов формируется список.

В чем разница между вызовами **`s.split()`** и **`s.split(' ')`**?

Метод `split`

Часто кажется, что метод `split()` одинаково работает с пустыми скобками и с пробелом, переданным в качестве аргумента. Но это не всегда так. Если в строке слова разделены более, чем одним пробелом, то в первом случае все пробелы пропадут и останутся только слова, а во втором пропадет только первый пробел в каждой группе, остальные останутся элементами списка:

```
line = "Whirling, twirling    round and   round"
print(line.split())
# ['Whirling,', 'twirling', 'round', 'and', 'round']
print(line.split(' '))
# ['Whirling,', 'twirling', '', '', '', 'round', 'and', '', '', 'round']
```

Метод `split`

Также неожиданно может повести себя этот метод, если слова в строке разделены разными пробельными символами. При использовании `split()` без аргумента слова будут разделены по любым пробельным символам, при конкретном указании — только по указанным. В следующем примере слово **to** отделено от других табуляцией, в остальных случаях оставлены пробелы.

```
line = "Falling softly	to	the ground"
print(line.split())
# ['Falling', 'softly', 'to', 'the', 'ground']
print(line.split(' '))
# ['Falling', 'softly', '', 'to', '', 'the', 'ground']
```

Метод `split`

Вот еще пример с разделением текста по переводу строки:

```
line = """Little leaves fall softly down
To make a carpet on the ground.
Then, swish, the wind comes whistling by
And sends them dancing to the sky.
"""
print(line.split())
print(line.split('\n'))
```

```
['Little', 'leaves', 'fall',
 'softly', 'down', 'To', 'make',
 'a', 'carpet', 'on', 'the', 'ground.',
 'Then,', 'swish,', 'the', 'wind',
 'comes', 'whistling', 'by',  'And',
 'sends', 'them', 'dancing',
 'to', 'the', 'sky.']
```

```
['Little leaves fall softly down',
 'To make a carpet on the ground.',
 'Then, swish, the wind comes whistling by',
 'And sends them dancing to the sky.',
 '']
```

Лайфхаки с методом **split**

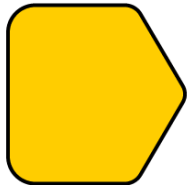
У метода *split* есть еще некоторые приятные возможности. У него дополнительный аргумент *maxsplit*. С его помощью можно не просто разбить строку по разделителю и превратить ее в список, но еще и указать, сколько раз считать разделители с начала строки.

```
name, lastname, *buy = 'Betty Botter bought a bit of butter'.split(maxsplit=2)
print(name, lastname, buy, sep='\n')
```

```
Betty
Botter
bought a bit of butter
```

С помощью метода *rsplit* можно разбивать строку с конца, а не с начала.

Метод `join`



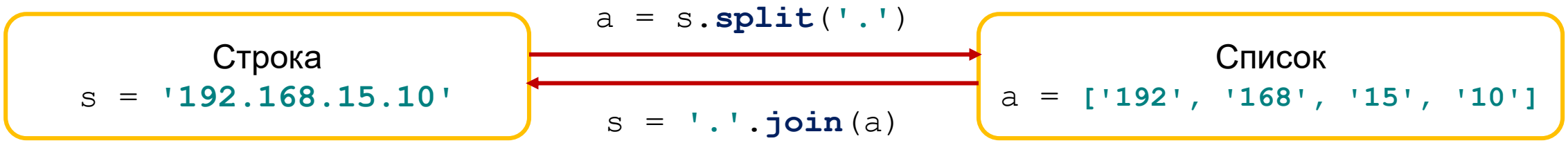
`join` же всегда принимает один аргумент — список слов, которые нужно **склеить**. Разделителем (точнее, «соединителем») служит та самая строка, чей метод `join` вызывается. Это может быть и пустая строка, и пробел, и символ новой строки, и что угодно ещё.

В этом примере все сравнения истинны, т. е. все вызовы функции `print` выведут `True`.

```
s = ['Тот', 'Кого', 'Нельзя', 'Называть']
print(''.join(s) == 'ТотКогоНельзяНазывать')
print(' '.join(s) == 'Тот Кого Нельзя Называть')
print('-'.join(s) == 'Тот-Кого-Нельзя-Называть')
print('! '.join(s) == 'Тот! Кого! Нельзя! Называть')
```

Методы `split` и `join`

Итак, `split` служит для преобразования строки в список,
а `join` — для преобразования списка в строку:



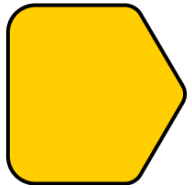
Обратите внимание, что `split` и `join` — это методы **строк**. Попытка вызвать такой метод у объекта, не являющегося строкой, вызовет ошибку! Например, если попытаться написать заведомо бессмысленное с точки зрения интерпретатора `Python` выражение:

```
[1, 2, 3].join([4, 5, 6])
--------------------------------------------------
AttributeError: 'list' object has no attribute 'join'
```

AttributeError

Списочные выражения

Списочные выражения



Для генерации списков из неповторяющихся элементов в **Python** имеется удобная синтаксическая конструкция — списочное выражение (**list comprehension**). Она позволяет создавать элементы списка в цикле **for**, не записывая цикл целиком.

Например, если нам необходимо создать список квадратов целых чисел от 0 до 9 включительно, мы можем записать следующий код:

```
squares = []
for i in range(10):
    squares.append(i ** 2)
print(squares)
```

То же самое, но гораздо короче, можно сделать с помощью списочного выражения:

```
squares = [i ** 2 for i in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

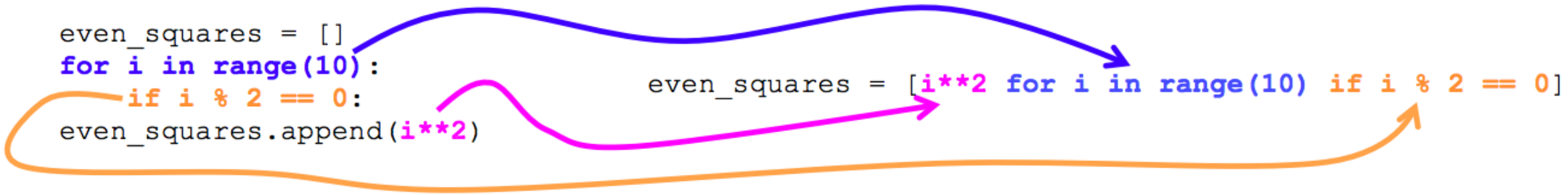
Списочные выражения

А если нам необходимы квадраты не всех чисел, а только чётных? Тогда можно добавить условие:

```
even_squares = []
for i in range(10):
    if i % 2 == 0:
        even_squares.append(i ** 2)
print(even_squares)
```

```
[0, 4, 16, 36, 64]
```

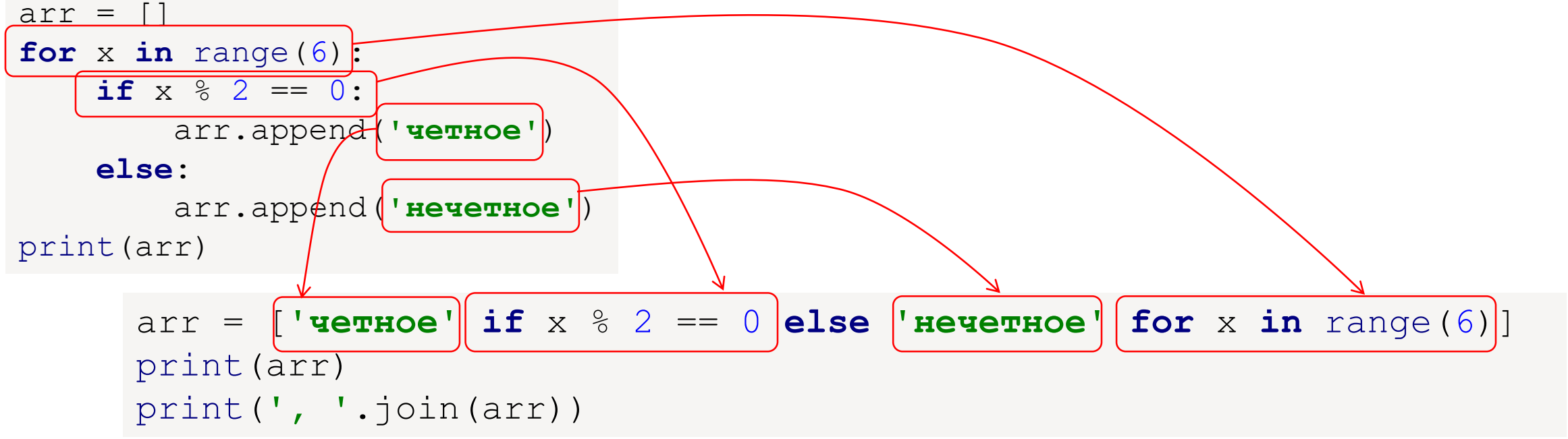
```
even_squares = [i ** 2 for i in range(10) if i % 2 == 0]
print(even_squares)
```



Списочные выражения `if - else`

В списочном выражении if пишется после цикла. Однако, иногда нужно выполнять разные действия при выполнении и невыполнении условия. В таком случае следует использовать тернарный условный оператор:

```
arr = []
for x in range(6):
    if x % 2 == 0:
        arr.append('четное')
    else:
        arr.append('нечетное')
print(arr)
```



```
arr = ['четное' if x % 2 == 0 else 'нечетное' for x in range(6)]
print(arr)
print(', '.join(arr))
```

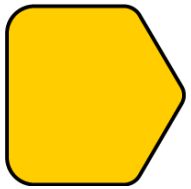
```
['четное', 'нечетное', 'четное', 'нечетное', 'четное', 'нечетное']
четное, нечетное, четное, нечетное, четное, нечетное
```

Списочные выражения

В списочном выражении можно пройти по двум или более циклам:

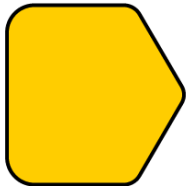
```
print([i * j for i in range(3) for j in range(3)])
```

```
[0, 0, 0, 0, 1, 2, 0, 2, 4]
```



На самом деле квадратные скобки не являются неотъемлемой частью списочного выражения. Если их не поставить, выражение будет вычисляться «по мере надобности» — когда очередной элемент становится нужен. "Заворачивая" списочное выражение в квадратные скобки, мы тем самым даем инструкцию сразу создать все элементы и составить из них список.

Списочные выражения



Списочные выражения часто используются для инициализации списков. Дело в том, что в **Python** не принято создавать пустые списки, чтобы потом заполнять их значениями, если можно этого избежать.

Если же все-таки необходимо создать пустой список, скажем, длиной 10, и заполнить его нулями (не может же он быть совсем пустой), то это легко сделать, используя умножение списка на число:

```
a = [0] * 10
```

Использование списочных выражений в аргументах методов split и join.

Использование списочных выражений в аргументах методов `split` и `join`.

Списочные выражения часто используют в аргументах методов **`split`** и **`join`**. Например, комбинация метода **`split`** и списочного выражения позволяют нам удобно считать числа, записанные в одну строку:

```
a = [int(x) for x in '976 929 289 809 7677'.split()]
evil, good = [int(x) for x in '666 777'.split()]
print(evil, good, sep='\n')
```

```
666
777
```

Здесь строка (обычно она не задаётся прямо в выражении, а получается из **`input()`**) разделяется на отдельные слова с помощью **`split`**. Затем списочное выражение пропускает каждый элемент получившегося списка через функцию **`int`**, превращая строку **`'976'`** в число **`976`**. Можно собрать все получившиеся значения в один список или разложить их по отдельным переменным с помощью множественного присваивания (как во второй строчке примера).

Использование списочных выражений в аргументах методов **split** и **join**.

Рассмотрим также пример использования метода **join** вместе со списочным выражением.
Выведем на одной строке список квадратов натуральных чисел от **1** до **9**:
1^2=1, 2^2=4, 3^2=9...
Для этого сначала с помощью списочного выражения сформируем список строк вида
['1^2=1', '2^2=4', '3^2=9', ...],
а затем "склеим" его в одну строку методом **join**:

```
print(', '.join(str(i) + '^2=' + str(i ** 2) for i in range(1, 10)))
```

```
1^2=1, 2^2=4, 3^2=9, 4^2=16, 5^2=25, 6^2=36, 7^2=49, 8^2=64, 9^2=81
```


Яндекс