Списочные выражения. Методы Split и Join

Методы Split и Join

`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

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(' ')`?

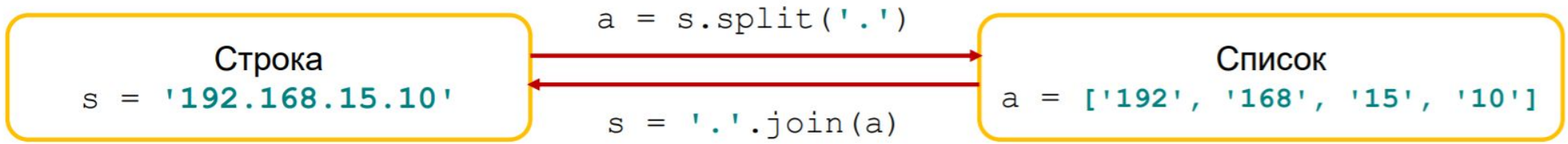
Метод Split

```
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 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.

```
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')
```




```
[1, 2, 3].join([4, 5, 6])
```

```
AttributeError: 'list' object has no attribute 'join'
```

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.

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

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

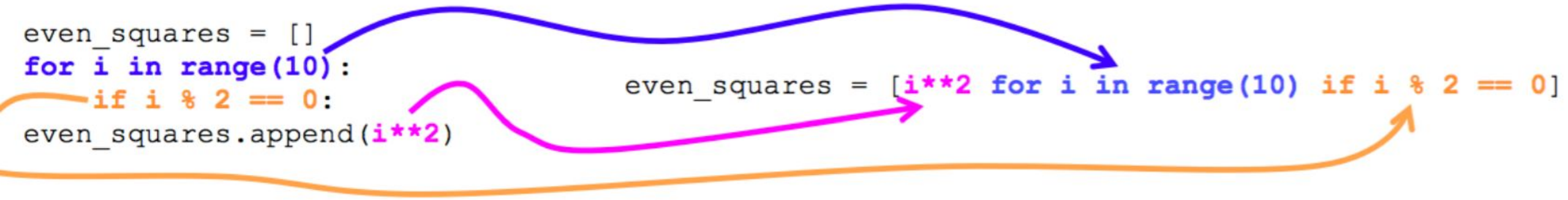
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Списочные выражения

```
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

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]
```

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.

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

Списочные выражения и аргументы 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).

Списочные выражения
и
аргументы Split и 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
```