* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/123/groups/568)

[Урок split и join](https://lyceum.yandex.ru/courses/123/groups/568/lessons/652)

**Методы split и join. Списочные выражения**

**План урока**

1

[Методы split и join](https://lyceum.yandex.ru/courses/123/groups/568/lessons/652/materials/1029#1)

2

[Списочные выражения](https://lyceum.yandex.ru/courses/123/groups/568/lessons/652/materials/1029#2)

3

[Использование списочных выражений в аргументах методов split и join. Считывание значений, введённых одной строкой](https://lyceum.yandex.ru/courses/123/groups/568/lessons/652/materials/1029#3)

**Аннотация**

*В этом уроке рассматриваются два очень популярных метода строк split и join. Во второй части рассматриваются списочные выражения (python list comprehensions [... for ... in ...]), которые в сочетании с методом split позволяют, например, считывать из стандартного потока ввода несколько значений одной строкой.*

**1. 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 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.

Для того, чтобы вызвать эти методы, необходимо использовать уже знакомый нам 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.

**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 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'])

**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 этих примерах также все сравнения истинны и каждый print выведет 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')

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 Traceback (most recent call last)

<ipython-input-3-404b5f4d9169> **in** <module>()

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

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

Как видно, указан тип объекта ([1, 2, 3] — список, list) и имя отсутствующего у него метода — join, который мы пытаемся вызвать.

**2. 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)

------------------------------------------------

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

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]

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]

То же самое, но короче, с помощью списочного выражения:

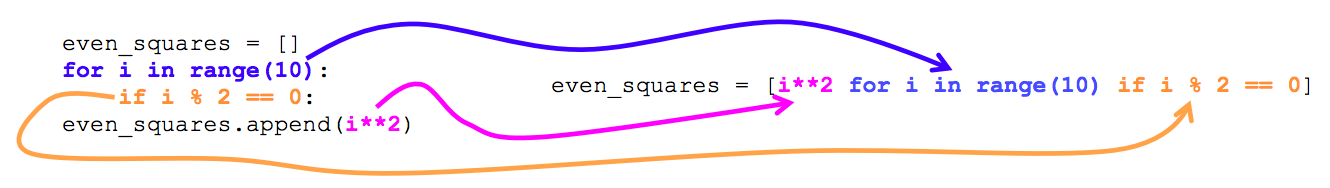
even\_squares = [i\*\*2 **for** i **in** range(10) **if** i % 2 == 0]

**print**(even\_squares)

------------------------------------------------

[0, 4, 16, 36, 64]

На рисунке показано соответствие между кодом заполнения списка с помощью цикла for и списочным выражением:



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. Пока что мы в основном будем пользоваться именно такими списочными выражениями — завёрнутыми в квадратные скобки и превращёнными тем самым в список.

**Важно**

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: **[0]\*10**.

**3. Использование списочных выражений в аргументах методов split и join. Считывание значений, введённых одной строкой**

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).

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

Заметьте, что в аргументе функции join стоит списочное выражение, не обёрнутое в квадратные скобки (это можно сделать, но не обязательно).

Будьте внимательны! Обычно возведение в степень обозначают «крышечкой» перед степенью, но в Python эта «крышечка» обозначает совсем другое, а возведение в степень выполняется оператором \*\*. В примере «крышечка» используется только как обозначение операции возведения в степень.

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты