Урок №11. Методы split и join. Списочные выражения.

План урока

1. Метод `split`.
2. Метод `join`.
3. Умножение списков и строк на целое число.
4. Списочные выражения.
5. Списочные выражения с условием.
6. Практика генерации списков и вложенных списков.
7. Считывание значений, введённых одной строкой.

Аннотация

*В первой части урока рассматривается понятие **метода** на примере популярных методов строки `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.*

Методы split и join

Мы уже встречались с методом `append` — функцией, «приклеенной» к списку и изменяющей его содержимое. Теперь изучим два других полезных метода строк: `join` и `split`. Они противоположны по смыслу: `split` разбивает строку по произвольному разделителю на список «слов», а `join` собирает из списка слов единую строку через произвольный разделитель.

Эти методы вызываются так же, как и метод `append`. 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 от `append`, не изменяют объект, которому принадлежат, а **создают новый** (список или строку, соответственно) и возвращают его, как это делают обычные функции типа `len`.

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'])
```

```
True
True
True
True
True
```

`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')
```

```
True
True
True
True
```

`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 Питона выражение:

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

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
<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.

Задача:

- **Глория Скотт (https://lms.yandexlyceum.ru/task/view/1246)**

Необязательная задача:

- **etc/passwd (https://lms.yandexlyceum.ru/task/view/1253)**

Списочные выражения

Мы уже знаем несколько простых способов создания строки. Изучим ещё один: строку из заданного количества повторяющихся подстрок можно легко составить путём умножения на число:

```
print("'#' * 5        -->", '#' * 5)  # выведет #####
print("'-|' * 4 + '-' -->", '-|' * 4 + '-')  # выведет -|-|-|-|-
```

```
'#' * 5        --> #####
'-|' * 4 + '-' --> -|-|-|-|-
```

Так же можно делать и для списков:

```
print([2, 3] * 4)
```

```
[2, 3, 2, 3, 2, 3, 2, 3]
```

Для генерации списков и строк, состоящих строго из повторяющихся элементов, умножение на число — лучший метод.

Задача:

- **Горизонтальная диаграмма (https://lms.yandexlyceum.ru/task/view/1247)**

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

```
squares = []
for i in range(10):
    squares.append(i**2)
print(squares)
# то же самое, но короче, с помощью списочного выражения:
squares = [i**2 for i in range(10)]
print(squares)
```

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

Можно даже добавить условие:

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

В списочном выражении можно пройти по двум или более циклам:

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

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

В действительности квадратные скобки не являются неотъемлемой частью списочного выражения. Это выражение выполняет вычисления «по мере надобности» — когда очередной элемент становится нужен. Когда мы заворачиваем списочное выражение в квадратные скобки, мы тем самым велим ему сразу создать все элементы и составить из них список. Пока мы в основном будем пользоваться именно такими списочными выражениями — завёрнутыми в квадратные скобки и превращёнными тем самым в список.

В Python не принято создавать пустые списки, чтобы потом заполнять их значениями, если можно этого избежать.

Но всё-таки, пусть необходимо создать пустой список — скажем, длиной 10 — и заполнить его нулями (не может же он быть совсем пустой). Это легко: `[0]*10`.

А если нужен список из 10 таких списков — двумерный список? Могло бы показаться, что сработает конструкция `a=[[0]*10]*10`, но это не так. Попробуйте понять, почему.

Подсказка: создайте такой список, измените в нём один элемент и посмотрите, что получилось.

Самый короткий способ выполнить такую задачу — при помощи списочного выражения:

```
[[0] * 10 for _ in range(10)]
```

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Напомним, что _ — законное имя переменной, как и, например, `i`. Однако по соглашению оно используется для переменной-счётчика: принимаемые этой переменной значения не важны, важно лишь количество итераций.

Наконец, метод `split` и списочные выражения позволяют нам удобно считать числа, записанные в 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).

Другой пример: с помощью `join` выводим на одной строке список квадратов натуральных чисел: `1^2=1`, `2^2=4`, `3^2=9`... 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.

```
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
```

Задачи

- Списочная квадратура (https://lms.yandexlyceum.ru/task/view/1248) +,
- Списочная квадратура — 2 (https://lms.yandexlyceum.ru/task/view/1249) +,
- Списочная квадратура — 3 (https://lms.yandexlyceum.ru/task/view/1250) +