Знакомство с map() в Python

**ОПУБЛИКОВАНО**[**07.04.2021**](https://pythonist.ru/python-map-znakomstvo/)[**ФУНКЦИИ**](https://pythonist.ru/category/voprosy-s-sobesedovaniya-python/funkczii/)**,**[**ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ**](https://pythonist.ru/category/functionalprogramming/)

Представляем вам первую часть перевода статьи [Python’s map(): Processing Iterables Without a Loop](https://realpython.com/python-map-function/), опубликованного [сайтом webdevblog.ru](https://webdevblog.ru/python-map-obrabotka-massivov-bez-ciklov/).

В первой части:

* [Функциональный стиль на Python](https://pythonist.ru/python-map-znakomstvo/?utm_source=telegram&utm_medium=pythonist#functional-style)
* [Начало работы с map в Python](https://pythonist.ru/python-map-znakomstvo/?utm_source=telegram&utm_medium=pythonist#getting-started-with-map)
  + [Что такое map()](https://pythonist.ru/python-map-znakomstvo/?utm_source=telegram&utm_medium=pythonist#understanding-map)
  + [Использование map() с различными видами функций](https://pythonist.ru/python-map-znakomstvo/?utm_source=telegram&utm_medium=pythonist#using-map-with-different-kinds-of-functions)
  + [Обработка множественных итераций с помощью map()](https://pythonist.ru/python-map-znakomstvo/?utm_source=telegram&utm_medium=pythonist#processing-multiple-input-iterables-with-map)

Python [**map()**](https://docs.python.org/3/library/functions.html#map) — это встроенная функция, которая позволяет обрабатывать и преобразовывать все элементы в итерируемом объекте без использования явного цикла for, методом, широко известным как сопоставление ([mapping](https://ru.wikipedia.org/wiki/Map_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))). **map()** полезен, когда вам нужно применить функцию преобразования к каждому элементу в коллекции или в массиве и преобразовать их в новый массив.

**map()** — один из инструментов, поддерживающих стиль функционального программирования в Python.

**Из этой статьи вы узнаете:**

* Как работает Python map()
* Как преобразовать различные типы массивов Python с помощью map()
* Как объединить map() с другими функциональными инструментами для выполнения более сложных преобразований
* Какие инструменты вы можете использовать, чтобы заменить map() и сделать свой код более Pythonic

Обладая этими знаниями, вы сможете эффективно использовать map() в своих программах или, в качестве альтернативы, использовать [**списковое включение**](https://pythonist.ru/kogda-ispolzovat-list-comprehension-v-python/) (list comprehensions) или **выражения-генераторы** (generator expressions), чтобы сделать ваш код более питоническим и читабельным.

Для лучшего понимания работы map() вам были бы полезны некоторые знания о том, как работать с итерациями (iterables), циклами, функциями (functions) и лямбда-функций (lambda functions).

**Функциональный стиль в Python**

В функциональном программировании вычисления выполняются путем объединения функций, которые принимают аргументы и возвращают конкретное значение (или значения). Эти функции не изменяют свои входные аргументы и не изменяют состояние программы. Они просто предоставляют результат данного вычисления. Такие функции обычно называются чистыми функциями ([pure functions](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D1%82%D0%BE%D1%82%D0%B0_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8)).

Теоретически программы, построенные с использованием функционального стиля, проще:

* **Разрабатывать,** потому что вы можете кодировать и использовать каждую функцию изолированно
* **Отлаживать и тестировать,**потому что вы можете тестировать и отлаживать отдельные функции, не глядя на остальную часть программы
* **Понимать**, потому что вам не нужно иметь дело с изменениями состояния на протяжении всей программы

Функциональное программирование обычно использует списки, массивы и другие итерационные объекты для представления данных вместе с набором функций, которые работают с этими данными и преобразовывают их. Когда дело доходит до обработки данных в функциональном стиле, обычно используются как минимум три метода:

1. Сопоставление ([**Mapping**](https://en.wikipedia.org/wiki/Map_(higher-order_function))) заключается в применении функции преобразования к итерируемому объекту для создания нового объекта. Элементы в новой итерации создаются путем вызова функции преобразования для каждого элемента в исходной итерации.
2. Фильтрация ([**Filtering**](https://en.wikipedia.org/wiki/Filter_(higher-order_function))) состоит из применения предиката или булевозначной функции ([predicate or Boolean-valued function](https://en.wikipedia.org/wiki/Boolean-valued_function)) к итерируемому объекту для создания нового итерируемого объекта. Элементы в новой итерации создаются путем фильтрации любых элементов в исходной итерации, которые заставляют функцию предиката возвращать **false**.
3. Сокращение ([**Reducing**](https://en.wikipedia.org/wiki/Fold_(higher-order_function))) состоит из применения функции reduce к итерируемому объекту для получения единственного накопленного значения.

По словам [Гвидо ван Россума](https://ru.wikipedia.org/wiki/%D0%92%D0%B0%D0%BD_%D0%A0%D0%BE%D1%81%D1%81%D1%83%D0%BC,_%D0%93%D0%B2%D0%B8%D0%B4%D0%BE), на Python в большей степени влияют императивные языки программирования, чем функциональные языки:

*Я никогда не считал, что Python находится под сильным влиянием функциональных языков, независимо от того, что люди говорят или думают. Я был более знаком с императивными языками, такими как C и Algol 68, и хотя я сделал функции первоклассными объектами (first-class objects), я не рассматривал Python как язык функционального программирования. (*[*Источник*](https://web.archive.org/web/20161104183819/http:/python-history.blogspot.com.br/2009/04/origins-of-pythons-functional-features.html)*)*

Однако еще в 1993 году сообщество Python требовало некоторых функций функционального программирования. Они просили:

* [Анонимные функции](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%BE%D0%BD%D0%B8%D0%BC%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F)
* Функцию map()
* Функцию  filter()
* Функцию  reduce()

Эти функциональные возможности были добавлены в язык благодаря участию многих членов сообщества. В настоящее время  [map()](https://docs.python.org/3/library/functions.html#map), [filter()](https://docs.python.org/3/library/functions.html#filter) и [reduce()](https://realpython.com/python-reduce-function/) являются фундаментальными компонентами стиля функционального программирования в Python.

В этом руководстве мы рассмотрим одну из этих функциональных возможностей — встроенную карту функций **map()**. Вы также узнаете, как использовать составные части списковых включений ([comprehensions](https://pythonist.ru/kogda-ispolzovat-list-comprehension-v-python/)) и выражения генератора ([generator expressions](https://pythonist.ru/generatory-python-ih-sozdanie-i-ispolzovanie/)), чтобы получить ту же функциональность, что и map(), в питоническом и удобочитаемом виде.

**Начало работы с map() в Python**

Иногда вы можете столкнуться с ситуациями, когда вам нужно выполнить одну и ту же операцию со всеми элементами массива, чтобы создать новый массив. Самый быстрый и распространенный подход к этой проблеме — использовать цикл **for**в Python. Однако вы также можете решить эту проблему без явного использования циклов, используя **map()**.

В следующих трех разделах вы узнаете, как работает **map()** и как вы можете использовать егр для обработки и преобразования итераций без циклов.

**Что такое map()**

**map()** перебирает элементы итерируемого массива (или коллекции) и возвращает новый массив (или итерируемый объект), который является результатом применения функции преобразования к каждому элементу исходного итерабельного массива.

Согласно [документации](https://docs.python.org/3/library/functions.html#map), map() принимает функцию и итерацию (или несколько итераций) в качестве аргументов и возвращает итератор, который выдает преобразованные элементы по запросу. Сигнатура функции map определяется следующим образом:

map(function, iterable[, iterable1, iterable2,..., iterableN])

**map()** применяет функцию к каждому элементу в итерируемом цикле и возвращает новый итератор, который по запросу возвращает преобразованные элементы. function может быть любая функция Python, которая принимает принимать аргументы, равное количеству итераций, которые вы передаете map().

**Примечание**. Первый аргумент map() — это объект функция, что означает, что вам нужно передать функцию, не вызывая ее. **То есть без пары скобок**.

Первый аргумент map() — **функция преобразования**. Другими словами, это функция, которая преобразует каждый исходный элемент в новый (преобразованный) элемент. Несмотря на то, что документация Python вызывает эту функцию аргумента, она может быть любой вызываемой Python. Сюда входят встроенные функции, классы, методы, лямбда-функции и пользовательские функции.

Операция, выполняемая map(), обычно известна как **сопоставление**, потому что она сопоставляет каждый элемент во входном итерируемом элементе с новым элементом в итоговом итерируемом. Для этого map() применяет функцию преобразования ко всем элементам во входной итерации.

Чтобы лучше понять map(), предположим, что вам нужно взять список числовых значений и преобразовать его в список, содержащий квадратное значение каждого числа в исходном списке. В этом случае вы можете использовать цикл **for**и написать что-то вроде этого:

>>> numbers = [1, 2, 3, 4, 5]

>>> squared = []

>>> **for** num **in** numbers:

... squared.append(num \*\* 2)

...

>>> squared

[1, 4, 9, 16, 25]

Когда вы запускаете этот цикл для чисел, вы получаете список квадратных значений. Цикл **for**перебирает числа и применяет к каждому значению операцию возведения в квадрат. Наконец, он сохраняет полученные значения в **squared**.

Вы можете добиться того же результата без использования явного цикла **for**, используя **map()**. Взгляните на следующую реализацию приведенного выше примера:

>>> **def** square(number):

... **return** number \*\* 2

...

>>> numbers = [1, 2, 3, 4, 5]

>>> squared = map(square, numbers)

>>> list(squared)

[1, 4, 9, 16, 25]

**square()** — это функция преобразования, которая преобразует число в его квадратное значение. Вызов **map()** применяет **square()** ко всем значениям и возвращает итератор, который возвращает квадратные значения. Затем вызывается **list()** для **map()**, чтобы создать объект списка, содержащий квадратные значения.

Поскольку **map()** написан на C и сильно оптимизирован, его внутренний подразумеваемый цикл может быть более эффективным, чем обычный цикл **for** в Python. Это одно из преимуществ использования **map()**.

Второе преимущество использования **map()** связано с потреблением памяти. С помощью цикла **for** вам нужно сохранить весь список в памяти вашей системы. С помощью **map()** вы получаете элементы по запросу, и только один элемент находится в памяти вашей системы в данный момент.

**Примечание**. В Python 2.x **map()** возвращает список. Это поведение изменилось в Python 3.x. Теперь map() возвращает объект **map**, который является итератором, выдающим элементы по запросу. Вот почему вам нужно вызвать **list()**, чтобы создать желаемый объект списка.

В качестве другого примера предположим, что вам нужно преобразовать все элементы в списке из строки в целое число. Для этого вы можете использовать map() вместе с int() следующим образом:

>>> str\_nums = ["4", "8", "6", "5", "3", "2", "8", "9", "2", "5"]

>>> int\_nums = map(int, str\_nums)

>>> int\_nums

<map object at 0x7fb2c7e34c70>

>>> list(int\_nums)

[4, 8, 6, 5, 3, 2, 8, 9, 2, 5]

>>> str\_nums

["4", "8", "6", "5", "3", "2", "8", "9", "2", "5"]

map() применяет [int()](https://docs.python.org/3/library/functions.html#int) к каждому значению в str\_nums. Поскольку map() возвращает итератор (объект map), вам понадобится вызов list(), чтобы вы могли превратить его в объект списка. Обратите внимание, что исходная последовательность не изменяется в процессе

**Использование map() с различными видами функций**

Вы можете использовать любую функцию Python, вызываемую с помощью map(). Единственным условием будет то, что вызываемый объект принимает аргумент и возвращает конкретное и полезное значение. Например, вы можете использовать классы, экземпляры, реализующие специальный метод с именем [**\_\_call\_\_**](https://docs.python.org/3/reference/datamodel.html#object.__call__), методы экземпляра, методы класса, статические методы и функции.

Есть несколько встроенных функций, которые вы можете использовать с map(). Рассмотрим следующие примеры:

>>> numbers = [-2, -1, 0, 1, 2]

>>> abs\_values = list(map(abs, numbers))

>>> abs\_values

[2, 1, 0, 1, 2]

>>> list(map(float, numbers))

[-2.0, -1.0, 0.0, 1.0, 2.0]

>>> words = ["Welcome", "to", "Real", "Python"]

>>> list(map(len, words))

[7, 2, 4, 6]

Вы можете использовать любую встроенную функцию с map() при условии, что функция принимает аргумент и возвращает значение.

Когда дело доходит до использования map(), вы обычно видите использование лямбда-функции в качестве первого аргумента. лямбда-функции удобны, когда вам нужно передать функцию на основе выражений в map(). Например, вы можете повторно реализовать пример квадратных значений с помощью лямбда-функции следующим образом:

>>> numbers = [1, 2, 3, 4, 5]

>>> squared = map(**lambda** num: num \*\* 2, numbers)

>>> list(squared)

[1, 4, 9, 16, 25]

лямбда-функции весьма полезны, когда дело доходит до использования map(). Они могут играть роль первого аргумента map(). Вы можете использовать лямбда-функции вместе с map () для быстрой обработки и преобразования ваших итераций.

**Обработка множественных итераций с помощью map()**

Если вы предоставляете несколько итераций для map(), тогда функция преобразования должна принимать столько аргументов, сколько итераций, которые вы передаете. Каждая итерация map() будет передавать одно значение из каждой итерации в качестве аргумента функции. Итерация останавливается в конце самой короткой итерации.

Рассмотрим следующий пример, в котором используется pow():

>>> first\_it = [1, 2, 3]

>>> second\_it = [4, 5, 6, 7]

>>> list(map(pow, first\_it, second\_it))

[1, 32, 729]

pow() принимает два аргумента, **x** и **y**, и возвращает **x** в степени **y**. На первой итерации **x** будет 1, **y** будет 4, а результат будет 1. Во второй итерации **x** будет 2, **y** будет 5, а результат будет 32, и так далее. Последняя итерация равна длине самой короткой итерации, которой в данном случае является **first\_it**.

Этот метод позволяет объединить две или более итерации числовых значений, используя различные виды математических операций. Вот несколько примеров, в которых лямбда-функции используются для выполнения различных математических операций с несколькими входными итерациями:

>>> list(map(**lambda** x, y: x - y, [2, 4, 6], [1, 3, 5]))

[1, 1, 1]

>>> list(map(**lambda** x, y, z: x + y + z, [2, 4], [1, 3], [7, 8]))

[10, 15]

В первом примере используется операция вычитания, чтобы объединить две итерации по три элемента в каждой. Во втором примере складывается значения трех итераций.