Функция filter

Функция filter принимает критерий отбора элементов, а затем сам список элементов. Возвращает она список из элементов, удовлетворяющих критерию.

Чтобы этой функцией воспользоваться, нужно сообщить функции filter критерий, который говорит, брать элемент в результирующий список или нет. Давайте напишем простую функцию, которая проверяет, что слово длиннее шести букв, и затем отберем с ее помощью длинные слова.

```
def is_word_long(word):
    return len(word) > 6


words = ['В', 'новом', 'списке', 'останутся', 'только', 'длинные',
'слова']
for word in filter(is_word_long, words):
    print(word)
# => останутся
# => длинные
```

С методом filter вам не нужно вручную создавать и заполнять список, достаточно указать условие отбора.

Итераторы

Если вы попробуете распечатать результат функции filter при помощи функции print (а не перебирая элементы по одному в цикле for), удивитесь: будет выведен не список, а специальный объект `"<filter object at 0x...>"`.

Он похож на список тем, что его можно перебирать циклом for, т. е. итерировать. Такие объекты называют **итераторами**.

Чтобы получить из итератора список, можно воспользоваться функцией list:

```
long_words = list(filter(is_word_long, words))
```

Описанный способ отфильтровать список пока далек от удобного, поскольку нам приходится заводить функцию для каждой проверки, что занимает две лишних строки кода. Для каждой такой маленькой функции приходится придумывать имя (и загромождать пространство имен).

Для того чтобы создавать такие короткие функции, «на один раз», в языке Python есть специальный синтаксис.

Лямбда-функции
Часто в качестве аргумента для функций высшего порядка мы хотим использовать совсем простую функцию. Причем нередко такая функция нужна в программе только в одном месте, поэтому ей необязательно даже иметь имя.

Лямбда-функции

Такие короткие безымянные (анонимные) функции можно создавать инструкцией `lambda <аргументы>: <выражение>`.

Такая инструкция создаст функцию, принимающую указанный список аргументов и возвращающую результат вычисления выражения.

В языке Python тело лямбда-функции имеет ровно одно выражение. Инструкция return подразумевается, писать ее не требуется, да и нельзя. Скобки вокруг аргументов не пишутся, аргументы от выражения отделяет двоеточие.

Теперь мы можем записать функцию, проверяющую длину слова, следующим образом:

```
lambda word: len(word) > 6
```

И список длинных слов теперь извлечь очень просто:

```
long_words = list(filter(lambda word: len(word) > 6, words))
```

Лямбда-функция — полноценная функция. Ее можно использовать в составе любых конструкций. Например, если вы хотите использовать ее несколько раз, но не хотите определять функцию с помощью def, вы можете присвоить созданную лямбда-функцию какой-либо переменной.

```
add = lambda x, y: x + y
add(3, 5) # => 8
add(1, add(2, 3)) # => 6
```

А теперь рассмотрим вариант, что вам нужно взять все элементы списка, но в программе уже стоит filter и вам не хочется его удалять. В этой ситуации вам поможет функция с особенно простым выражением:

```
lambda x: True
```

Покажем:

```
def print_some_primes(criterion):
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
    for number in filter(criterion, primes):
        print(number)
```

```
print_some_primes(lambda x: True)
# => [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Важно!

Лямбда-функция принимает аргумент, хотя и не использует его. Функция filter всегда передает в критерий элемент, который проверяет. Если бы мы написали лямбда-функцию без аргументов `lambda: True`, функция filter вызвала бы ошибку, потому что передала бы аргумент в функцию, которая аргументов не принимает.

Чтобы функция filter не казалась магической, напишем свой упрощенный аналог. Наша функция simple_filter будет принимать критерий и список и возвращать новый список.

```
def simple_filter(criterion, arr):
    result = []
    for element in arr:
        if criterion(element):
            result.append(element)
    return result
```

Мы передали критерий как функцию, а потому можем его вызвать, что мы и сделали в условном операторе. Так как для перечисления элементов мы использовали конструкцию for, мы можем вместо списка в функцию передать любой итерируемый объект. Например, строку (элементами будут отдельные символы) или интервал range.

Например, найдем все числа от 1 до 99, которые при делении на 12 дают 7 в остатке.

```
simple_filter(lambda x: x % 12 == 7, range(1, 100))
# => [7, 19, 31, 43, 55, 67, 79, 91]
```

Функция map
Другая популярная функция высшего порядка называется map.

Функция map

Функция map преобразует каждый элемент списка по некоторому общему правилу и в результате создает список (вернее, как и filter, специальный итерируемый объект, похожий на список) из преобразованных значений.

Функция, которую map принимает, — преобразование одного элемента. Зная, как преобразуется один элемент, map выполняет превращение целых списков.

Например, возьмем набор из чисел от 1 до 10 и применим к ним функцию возведения в квадрат.

```
list(map(lambda x: x ** 2, range(1, 10)))
# => [1, 4, 9, 16, 25, 36, 49, 64, 81]
```