

Chapter 26: Filter

Parameter	Details
function	<i>callable</i> that determines the condition or <code>None</code> then use the identity function for filtering (<i>positional-only</i>)
iterable	iterable that will be filtered (<i>positional-only</i>)

Section 26.1: Basic use of filter

To `filter` discards elements of a sequence based on some criteria:

```
names = ['Fred', 'Wilma', 'Barney']

def long_name(name):
    return len(name) > 5

Python 2.x Version ≥ 2.0
filter(long_name, names)
# Out: ['Barney']

[name for name in names if len(name) > 5] # equivalent list comprehension
# Out: ['Barney']
```

```
from itertools import ifilter
ifilter(long_name, names)      # as generator (similar to python 3.x filter builtin)
# Out: <itertools.ifilter at 0x4197e10>
list(ifilter(long_name, names)) # equivalent to filter with lists
# Out: ['Barney']

(name for name in names if len(name) > 5) # equivalent generator expression
# Out: <generator object <genexpr> at 0x000000003FD5D38>
```

```
Python 2.x Version ≥ 2.6
# Besides the options for older python 2.x versions there is a future_builtin function:
from future_builtins import filter
filter(long_name, names)      # identical to itertools.ifilter
# Out: <itertools.ifilter at 0x3eb0ba8>
```

```
Python 3.x Version ≥ 3.0
filter(long_name, names)      # returns a generator
# Out: <filter at 0x1fc6e443470>
list(filter(long_name, names)) # cast to list
# Out: ['Barney']

(name for name in names if len(name) > 5) # equivalent generator expression
# Out: <generator object <genexpr> at 0x000001C6F49BF4C0>
```

Section 26.2: Filter without function

If the function parameter is `None`, then the identity function will be used:

```
list(filter(None, [1, 0, 2, [], '', 'a'])) # discards 0, [] and ''
# Out: [1, 2, 'a']

Python 2.x Version ≥ 2.0.1
[i for i in [1, 0, 2, [], '', 'a'] if i] # equivalent list comprehension

Python 3.x Version ≥ 3.0.0
```

```
(i for i in [1, 0, 2, [], '', 'a'] if i) # equivalent generator expression
```

Section 26.3: Filter as short-circuit check

`filter` (python 3.x) and `ifilter` (python 2.x) return a generator so they can be very handy when creating a short-circuit test like or or `and`:

Python 2.x Version ≥ 2.0.1

```
# not recommended in real use but keeps the example short:  
from itertools import ifilter as filter
```

Python 2.x Version ≥ 2.6.1

```
from future_builtins import filter
```

To find the first element that is smaller than 100:

```
car_shop = [('Toyota', 1000), ('rectangular tire', 80), ('Porsche', 5000)]  
def find_something_smaller_than(name_value_tuple):  
    print('Check {0}, {1}$'.format(*name_value_tuple))  
    return name_value_tuple[1] < 100  
next(filter(find_something_smaller_than, car_shop))  
# Print: Check Toyota, 1000$  
#         Check rectangular tire, 80$  
# Out: ('rectangular tire', 80)
```

The `next`-function gives the next (in this case first) element of and is therefore the reason why it's short-circuit.

Section 26.4: Complementary function: `filterfalse`, `ifilterfalse`

There is a complementary function for `filter` in the `itertools`-module:

Python 2.x Version ≥ 2.0.1

```
# not recommended in real use but keeps the example valid for python 2.x and python 3.x  
from itertools import ifilterfalse as filterfalse
```

Python 3.x Version ≥ 3.0.0

```
from itertools import filterfalse
```

which works exactly like the `generator filter` but keeps only the elements that are `False`:

```
# Usage without function (None):  
list(filterfalse(None, [1, 0, 2, [], '', 'a'])) # discards 1, 2, 'a'  
# Out: [0, [], '']
```

```
# Usage with function  
names = ['Fred', 'Wilma', 'Barney']  
  
def long_name(name):  
    return len(name) > 5  
  
list(filterfalse(long_name, names))  
# Out: ['Fred', 'Wilma']
```

```
# Short-circuit usage with next:  
car_shop = [('Toyota', 1000), ('rectangular tire', 80), ('Porsche', 5000)]  
def find_something_smaller_than(name_value_tuple):
```

```
print('Check {0}, {1}$'.format(*name_value_tuple)
return name_value_tuple[1] < 100
next(filterfalse(find_something_smaller_than, car_shop))
# Print: Check Toyota, 1000$
# Out: ('Toyota', 1000)
```

```
# Using an equivalent generator:
car_shop = [('Toyota', 1000), ('rectangular tire', 80), ('Porsche', 5000)]
generator = (car for car in car_shop if not car[1] < 100)
next(generator)
```