

---

# **pyiterable Documentation**

***Release 0.1.0***

**neverendingqs**

September 07, 2015



<b>1</b>	<b>Classes</b>	<b>3</b>
1.1	Iterable . . . . .	3
<b>2</b>	<b>Details</b>	<b>9</b>
<b>3</b>	<b>Release</b>	<b>11</b>
<b>4</b>	<b>Indices and tables</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>



Python comes with some nice built-in methods for operating on iterables, but it can get messy really quickly if you want to transform an iterable multiple times. Write more expressive code by chaining built-in transformations with this module.

Scroll down for examples.



## 1.1 Iterable

`class pyiterable.Iterable(iterable)`

**all()**

Equivalent to the built-in function **all()** (*iterable*)

**Returns** True if all elements in *Iterable* are True, else False

```
>>> Iterable([True, False, True]).all()
False
>>> Iterable([True, True, True, True]).all()
True
```

**any()**

Equivalent to the built-in function **any()** (*iterable*)

**Returns** True if any element in *Iterable* is True, else False

```
>>> Iterable([True, False, True]).any()
True
>>> Iterable([False, False, False, False]).any()
False
```

**enumerate** (*start=0*)

Equivalent to the built-in function **enumerate()** (*sequence*, *start=0*)

**Parameters** **start** – integer value to start from

**Returns** (**index + start**, **value**), where **sequence[index] == value**

```
>>> grades = Iterable(['a', 'b', 'c', 'd', 'f'])
>>> grades.enumerate().to_list()
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (5, 'f')]
>>> grades.enumerate(start=5).to_list()
[(5, 'a'), (6, 'b'), (7, 'c'), (8, 'd'), (5, 'f')]
```

**filter** (*function*)

Equivalent to the built-in function **filter()** (*function*, *iterable*)

**Parameters** **function** – function that returns **False** for items to exclude

**Returns** *Iterable* object that only contains items filtered by *function*

```
>>> grades = Iterable(['a', 'b', 'c', 'd', 'f'])
>>> grades.enumerate().filter(lambda i_x: i_x[0] < 3).to_list()
[(0, 'a'), (1, 'b'), (2, 'c')]
```

**first** (*function=None, default=None*)

Equivalent to calling `next( iter( filter(function, iterable) ), default )`

**Parameters**

- **function** – keyword-only; function used to filter unwanted values
- **default** – keyword-only value to return if *Iterable* is empty after filtered by *func*

**Returns** first value of *Iterable* filtered by *func*

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.first()
1
>>> values.first(function=lambda x: x > 5)
9
>>> values.first(function=lambda x: x > 10, default=0)
0
```

**len** ()

Equivalent to the built-in function `len( s )`

**Returns** number of items in *Iterable*

```
>>> grades = Iterable(['a', 'b', 'c', 'd', 'f'])
>>> grades.len()
5
```

**map** (*function*)

Equivalent to the built-in function `map(function, iterable)`

**Parameters** **function** – function applied to every item in *Iterable*

**Returns** *Iterable* of results

```
>>> numbers = Iterable([1, 3, 10, 4, 8])
>>> numbers.map(lambda x: x * 2).to_list()
[2, 6, 20, 8, 16]
```

**mapmany** (*function*)

Equivalent to calling `itertools.chain.from_iterable( map(function, iterable) )`

```
>>> values = Iterable([1, 2, 5, 9])
>>> func = lambda x: [x, x]
>>> values.map(func).to_list()
[[1, 1], [2, 2], [5, 5], [9, 9]]
>>> values.mapmany(func).to_list()
[1, 1, 2, 2, 5, 5, 9, 9]
```

**Parameters** **function** – function to be applied to each input, and outputs an iterable

**Returns** *Iterable* comprised of every element returned by **function**

**max** (*\*\*kwargs*)

Equivalent to the built-in function `max( iterable, *, key, default )`

**Parameters**



- **key** – keyword-only; function that returns the value to compare
- **default** – keyword-only; value to return if *Iterable* is empty. Only available in Python 3.4 or later

**Returns** largest item in *Iterable*

```
>>> grades = Iterable([('Charlie', 79), ('Alice', 94), ('Bob', 65)])
>>> grades.max(key=lambda x: x[1])
('Alice', 94)
```

**min**(\*\*kwargs)

Equivalent to the built-in function **min**( *iterable*, \*[ *key*, *default*] )

**Parameters**

- **key** – keyword-only; function that returns the value to compare
- **default** – keyword-only; value to return if *Iterable* is empty. Only available in Python 3.4 or later

**Returns** smallest item in *Iterable*

```
>>> grades = Iterable([('Charlie', 79), ('Alice', 94), ('Bob', 65)])
>>> grades.min(key=lambda x: x[1])
('Bob', 65)
```

**reduce**( *function*, *initializer=None* )

Equivalent to:

- **Python 2.x:** the built-in function **reduce**( *function*, *iterable*[, *initializer*] )
- **Python 3.x:** **reduce**( *function*, *iterable*[, *initializer*] ) in *functools*

Repeatedly applies *function* to sequence until one value is left

**Parameters**

- **function** – function that takes two values and returns a single value
- **initializer** – initial value combined with the first value in *Iterable*

**Returns** single value

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.reduce(lambda a, b: a + b)
17
>>> values.reduce(lambda a, b: a + b, 10)
27
```

**reversed**()

Equivalent to the built-in function **reversed**( *seq* )

**Returns** *Iterable* in the reversed order

```
>>> names = Iterable(['Bob', 'Alice', 'Daniel', 'Charlie'])
>>> names.reversed().to_list()
['Charlie', 'Daniel', 'Alice', 'Bob']
```

**sorted**(\*\*kwargs)

Equivalent to the built-in function **sorted**( *iterable*[, *cmp*[, *key*[, *reverse*]]] )

**Parameters**

- **cmp** – keyword-only; custom comparison function. Only available in Python 2.x

- **key** – keyword-only; function that returns the value to compare
- **reverse** – keyword-only; boolean; if True, *Iterable* is sorted with the largest value first

**Returns** a sorted *Iterable*

```
>>> grades = Iterable([('Charlie', 79), ('Alice', 94), ('Bob', 65)])
>>> grades.sorted().to_list()
[('Alice', 94), ('Bob', 65), ('Charlie', 79)]
>>> grades.sorted(key=lambda x: x[1]).to_list()
[('Bob', 65), ('Charlie', 79), ('Alice', 94)]
>>> grades.sorted(key=lambda x: x[1], reverse=True).to_list()
[('Alice', 94), ('Charlie', 79), ('Bob', 65)]
```

**sum**(start=0)

Equivalent to the built-in function `sum( iterable[, start] )`

**Parameters** **start** – starting value; default is 0

**Returns** sum of all values in *Iterable*

```
>>> numbers = Iterable([1, 3, 10, 4, 8])
>>> numbers.sum()
26
>>> numbers.sum(10)
36
```

**to\_list**()

Equivalent to the built-in function `list( iterable )`

**Returns** list

```
>>> grades = Iterable([('Alice', 94), ('Bob', 65), ('Charlie', 79), ('Daniel', 70)])
>>> grades
<pyiterable.iterable.Iterable object at 0x017BACB0>
>>> grades.to_list()
[('Alice', 94), ('Bob', 65), ('Charlie', 79), ('Daniel', 70)]
```

**to\_set**()

Equivalent to the built-in function `set( iterable )`

**Returns** set

```
>>> numbers = Iterable([10, 7, 28, 7, 19, 19, 70])
>>> numbers
<pyiterable.iterable.Iterable object at 0x017BA610>
>>> numbers.to_set()
{10, 19, 28, 70, 7}
```

**to\_tuple**()

Equivalent to the built-in function `tuple( iterable )`

**Returns** tuple

```
>>> numbers = Iterable([10, 7, 28, 7, 19, 19, 70])
>>> numbers
<pyiterable.iterable.Iterable object at 0x0130FE70>
>>> numbers.to_tuple()
(10, 7, 28, 7, 19, 19, 70)
```

**zip**(\*args)

Equivalent to the built-in function `zip( [iterable, ...] )`

**Parameters** *args* – any number of iterable objects

**Returns** list of tuples; i-th tuple contains all elements from each i-th element in *Iterable* and *\*args*

```
>>> left = Iterable(['Alice', 'Bob', 'Charlie', 'Daniel'])
>>> left.zip([94, 65, 79, 70]).to_list()
[('Alice', 94), ('Bob', 65), ('Charlie', 79), ('Daniel', 70)]
```



---

## Details

---

Inspired by:

- C#'s Enumerable class
- Apache Spark RDD Operations
- Java stream package

Instead of:

```
values = ["1", "2", "5", "9"]

to_int = map(lambda x: int(x), values)
filtered = filter(lambda x: x > 4)
sum = reduce(lambda a, b: a + b, to_int)
```

or:

```
values = ["1", "2", "5", "9"]

sum = reduce(
    lambda a, b: a + b,
    filter(
        lambda x: x > 4,
        map(lambda x: int(x), values)
    )
)
```

do this:

```
from pyiterable import Iterable
...
values = Iterable(["1", "2", "5", "9"])

sum = (values
      .map(lambda x: int(x))
      .filter(lambda x: x > 4)
      .reduce(lambda a, b: a + b)
)
```



---

### Release

---

#### 0.2.0

- Added `first()`, which gives you the first value in `Iterable`, with an optional default if no values exist
- Added `mapmany()`, which functions like `map`, except it expects more than one output for each item of `Iterable`

#### 0.1.0

- First release!
- *Iterable* class with equivalent built-in functions related to iterables





---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



**p**

`pyiterable`, 3



## A

`all()` (pyiterable.Iterable method), 3  
`any()` (pyiterable.Iterable method), 3

## E

`enumerate()` (pyiterable.Iterable method), 3

## F

`filter()` (pyiterable.Iterable method), 3  
`first()` (pyiterable.Iterable method), 4

## I

Iterable (class in pyiterable), 3

## L

`len()` (pyiterable.Iterable method), 4

## M

`map()` (pyiterable.Iterable method), 4  
`mapmany()` (pyiterable.Iterable method), 4  
`max()` (pyiterable.Iterable method), 4  
`min()` (pyiterable.Iterable method), 5

## P

pyiterable (module), 3

## R

`reduce()` (pyiterable.Iterable method), 5  
`reversed()` (pyiterable.Iterable method), 5

## S

`sorted()` (pyiterable.Iterable method), 5  
`sum()` (pyiterable.Iterable method), 6

## T

`to_list()` (pyiterable.Iterable method), 6  
`to_set()` (pyiterable.Iterable method), 6  
`to_tuple()` (pyiterable.Iterable method), 6

## Z

`zip()` (pyiterable.Iterable method), 6