pyiterable Documentation

Release 0.1.0

neverendingqs

June 18, 2016

Contents

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.

The module is available on PyPI via `pip`:

```
pip install pyiterable
```

Examples below.

CHAPTER 1

Classes

1.1 Iterable

class pyiterable.**Iterable**(*iterable*)

all()

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

Returns True if all elements in *self* 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 *self* is True, else False

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

concat(*iterable*)

Equivalent to calling **list(** *left* **) + list(** *right* **)**

Parameters **iterable** – iterable to concat with *self*

Returns New *Iterable* containing the elements from *self* and *iterable*

```
>>> left = [2, 10, 2, 2, 5, 9, 10]
>>> right = [13, -5, 1982, -10, 2384, 1982, 98]
>>> Iterable(left).concat(right).to_list()
[2, 10, 2, 2, 5, 9, 10, 13, -5, 1982, -10, 2384, 1982, 98]
```

contains(*value*)

Equivalent to calling **value in** *iterable*

Parameters **value** – value to search for inside *iterable*

Returns *True* if value exists inside *iterable*, otherwise false

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.contains(2)
True
>>> values.contains(4)
False
```

difference(*iterable*)

Equivalent to calling **set(** *left* **).difference(set (** *iterable* **))**

Parameters **iterable** – iterable to check against for differences

Returns New *Iterable* containing elements found in *self* but not *iterable*

```
>>> left = [2, 10, 1982, -5, 9, 10]
>>> right = [1982, -10, -5, 1982, 98]
>>> Iterable(left).difference(right).to_list()
[9, 2, 10]
```

distinct()

Equivalent to calling **set(** *iterable* **)**

Returns New *Iterable* containing only the distinct elements; order not preserved

```
>>> values = Iterable([2, 10, 2, 2, 5, 9, 10])
>>> values.distinct().to_list()
[9, 2, 10, 5]
```

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'), (9, '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(*filter_by=None*, *default=None*, *function=None*)

Equivalent to calling **next(iter(filter(** *filter_by, iterable* **)) ,** *default* **)**

Parameters

- **filter_by** – keyword-only; function used to filter unwanted values
- **default** – keyword-only; value to return if *self* is empty after filtered by *filter_by*
- **function** – deprecated; use *filter_by*

Returns first value of *self* filtered by *filter_by*

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.first()
1
>>> values.first(filter_by=lambda x: x > 5)
9
>>> values.first(filter_by=lambda x: x > 10) # Returns None
>>> values.first(filter_by=lambda x: x > 10, default=0)
0
```

get (*index*)

Equivalent to calling **list(** *iterable* **)[** *index* **]**

- This function will convert the *iterable* to a sequence type before retrieving the value at *index*
- *-1* is not supported to get the last element; use **last()** instead

Parameters **index** – element number inside *iterable*

Returns value at *index* from *iterable*

Raises **IndexError** – *index* is less than 0 or is out of bounds

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.get(2)
5
>>> values.get(-1)
IndexError: index out of range
>>> values.get(5)
IndexError: index out of range
```

intersection (*iterable*)

Equivalent to calling **set(** *left* **).intersection(set(** *right* **))**

Parameters **iterable** – iterable to intersect with *self*

Returns *Iterable* with distinct values found in both *self* and *iterable*

```
>>> left = [2, 10, 1982, -5, 9, 10]
>>> right = [1982, -10, -5, 1982, 98]
>>> Iterable(left).intersection(right).to_list()
[-5, 1982]
```

is_empty ()

Equivalent to calling **len(list(** *iterable* **)) == 0**

Returns *True* if *iterable* does not contain any elements; otherwise *False*

```
>>> Iterable([1, 2, 5, 9]).is_empty()
False
>>> Iterable([]).is_empty()
True
```

last (*filter_by=None*, *default=None*)

Equivalent to calling **next(iter(reversed(list(filter(** *filter_by, iterable* **))))) ,** *default* **)**

Parameters

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

Returns last value of *self* filtered by *filter_by*

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.last()
9
>>> values.last(filter_by=lambda x: x < 5)
2
>>> values.last(filter_by=lambda x: x < 1) # Returns None
>>> values.last(filter_by=lambda x: x < 1, default=0)
0
```

len()

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

Returns number of items in *self*

```
>>> 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 *self*

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* **))**

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

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

```
>>> 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]
```

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 *self* is empty. Only available in Python 3.4 or later

Returns largest item in *self*

```
>>> 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 *self* is empty. Only available in Python 3.4 or later

Returns smallest item in *self*

```
>>> 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 *self*

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 *self* in the reversed order

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

single(*filter_by=None*, *default=None*)

Equivalent to calling **first()**, except it raises *ValueError* if *iterable* contains more than one element

Parameters

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

Returns value of *self* filtered by *filter_by*

Raises **ValueError** – *iterable* contains more than one element after being filtered by *filter_by*

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.single()
ValueError: iterable [1, 2, 5, 9] contains more than one element
>>> values.single(filter_by=lambda x: x > 1)
ValueError: iterable [2, 5, 9] contains more than one element
>>> values.single(filter_by=lambda x: x > 5)
9
>>> values.single(filter_by=lambda x: x > 10) # Returns None
```

```
>>> values.single(filter_by=lambda x: x > 10, default=0)
0
```

skip(*count*)

Skips the first *count* elements in *iterable*

- This function will convert the *iterable* to a sequence type before retrieving the values
- If *count* is equal to or greater than the length of *iterable*, no elements are taken

Parameters **count** – number of values to skip

Returns *Iterable* containing all the elements of *iterable* without the first *count* elements

Raises **ValueError** – *count* is a negative value

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.skip(1).to_list()
[2, 5, 9]
>>> values.skip(3).to_list()
[9]
>>> values.skip(10).to_list()
[]
>>> values.take(-1).to_list()
ValueError: 'count' must be greater than 0
```

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, *self* 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
```

symmetric_difference(*iterable*)

Equivalent to calling **set(** *left* **).symmetric_difference(set(** *right* **))**

Parameters **iterable** – iterable to perform symmetric difference against

Returns *Iterable* with distinct values found in either *self* or *iterable* but not both

```
>>> left = [2, 10, 1982, -5, 9, 10]
>>> right = [1982, -10, -5, 1982, 98]
>>> Iterable(left).symmetric_difference(right).to_list()
[98, 2, 9, 10, -10]
```

take(*count*)

Gets the first *count* elements in *iterable*

- This function will convert the *iterable* to a sequence type before retrieving the values
- If *count* is equal to or greater than the length of *iterable*, all elements are taken

Parameters **count** – number of values to retrieve

Returns *Iterable* comprised of the first *count* elements

Raises **ValueError** – *count* is a negative value

```
>>> values = Iterable([1, 2, 5, 9])
>>> values.take(1).to_list()
[1]
>>> values.take(3).to_list()
[1, 2, 5]
>>> values.take(10).to_list()
[1, 2, 5, 9]
>>> values.take(-1).to_list()
ValueError: 'count' must be greater than 0
```

to_frozenset()

Equivalent to the built-in type **frozenset(** *iterable* **)**

Returns frozenset

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

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)
```

union(*iterable*)
Equivalent to calling **set(** *left* **).union(set(** *right* **))**

Parameters **iterable** – iterable to union with *self*

Returns *Iterable* with distinct values in either *self* or *iterable*

```
>>> left = [2, 10, 2, 2, 5, 9, 10]
>>> right = [1982, -10, 5, 1982, 9]
>>> Iterable(left).union(right).to_list()
[2, 5, 9, 10, -10, 1982]
```

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 *self* and **args*

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

CHAPTER 2

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)
)
```

CHAPTER 3

Release

0.4.0

- Bug fix; new `Iterable` objects should no longer mutate if the `iterable` passed into the constructor is mutated
- Added more sequence-like functionality: `get()`, `last()`, `skip()`, and `take()`
- Added `contains()`, `single()`, and `is_empty()`
- Added `filter_by` keyword parameter to replace `function` for `first()`, as `filter_by` is more self-explanatory

0.3.1

- Added support for Python 3.5
- Removed support for Python 3.2

0.3.0

- Added set-like functionality, including `difference()`, `intersection()`, `symmetric_difference()`, and `union()`.
- Added `concat()` as an alternative to `union()`
- Added `distinct()`
- Added frozenset support (`to_frozenset()`)

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

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

p