ScalaFunctional Documentation

Release 0.1.6

Pedro Rodriguez

July 22, 2015

Contents

Contents:

CHAPTER 1

functional package

1.1 Submodules

1.2 functional.chain module

class functional.chain.**FunctionalSequence**(*sequence*, *transform=None*)
Bases: object

FunctionalSequence is a wrapper around any type of sequence which provides access to common functional transformations and reductions in a data pipelining style

all()
Returns True if the truth value of all items in the sequence true.

```
>>> seq([True, True]).all()
True
```

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

Returns True if all items truth value evaluates to True

any()
Returns True if any element in the sequence has truth value True

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

```
>>> seq([False, False]).any()
False
```

Returns True if any element is True

cache()

count(*func*)
Counts the number of elements in the sequence which satisfy the predicate f.

```
>>> seq([-1, -2, 1, 2]).count(lambda x: x > 0)
2
```

Parameters **func** – predicate to count elements on

Returns count of elements that satisfy predicate

dict()
Converts sequence of (Key, Value) pairs to a dictionary.

```
>>> type(seq([('a', 1)]).to_dict())
dict
```

```
>>> type(seq([]))
functional.chain.FunctionalSequence
```

```
>>> seq([('a', 1), ('b', 2)]).to_dict()
{'a': 1, 'b': 2}
```

Returns dictionary from sequence of (Key, Value) elements

difference(*other*)
New sequence with unique elements present in sequence but not in other.

```
>>> seq([1, 2, 3]).difference([2, 3, 4])
[1]
```

Parameters **other** – sequence to perform difference with

Returns difference of sequence and other

:rtype FunctionalSequence

distinct()
Returns sequence of distinct elements. Elements must be hashable.

```
>>> seq([1, 1, 2, 3, 3, 3, 4]).distinct()
[1, 2, 3, 4]
```

Returns sequence of distinct elements

:rtype FunctionalSequence

distinct_by(*func*)
Returns sequence of elements who are distinct by the passed function. The return value of func must be hashable. When two elements are distinct by func, the first is taken. :param func: function to use for determining distinctness :return: elements distinct by func :rtype FunctionalSequence

drop(*n*)
Drop the first n elements of the sequence.

```
>>> seq([1, 2, 3, 4, 5]).drop(2)
[3, 4, 5]
```

Parameters **n** – number of elements to drop

Returns sequence without first n elements

:rtype FunctionalSequence

drop_right(*n*)
Drops the last n elements of the sequence.

```
>>> seq([1, 2, 3, 4, 5]).drop_right(2)
[1, 2, 3]
```

Parameters **n** – number of elements to drop

Returns sequence with last n elements dropped

:rtype FunctionalSequence

drop_while(*func*)

Drops elements in the sequence while f evaluates to True, then returns the rest.

```
>>> seq([1, 2, 3, 4, 5, 1, 2]).drop_while(lambda x: x < 3)
[3, 4, 5, 1, 2]
```

Parameters **func** – truth returning function

Returns elements including and after f evaluates to False

:rtype FunctionalSequence

empty()

Returns True if the sequence has length zero.

```
>>> seq([]).empty()
True
```

```
>>> seq([1]).empty()
False
```

Returns True if sequence length is zero

enumerate(*start=0*)

Uses python enumerate to to zip the sequence with indexes starting at start.

```
>>> seq(['a', 'b', 'c']).enumerate(start=1)
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Parameters **start** – Beginning of zip

Returns enumerated sequence starting at start

:rtype FunctionalSequence

exists(*func*)

Returns True if an element in the sequence makes f evaluate to True.

```
>>> seq([1, 2, 3, 4]).exists(lambda x: x == 2)
True
```

```
>>> seq([1, 2, 3, 4]).exists(lambda x: x < 0)
False
```

Parameters **func** – existence check function

Returns True if any element satisfies f

filter (*func*)
Filters sequence to include only elements where f is True.

```
>>> seq([-1, 1, -2, 2]).filter(lambda x: x > 0)
[1, 2]
```

Parameters **func** – function to filter on

Returns filtered sequence

:rtype FunctionalSequence

filter_not (*func*)
Filters sequence to include only elements where f is False.

```
>>> seq([-1, 1, -2, 2]).filter_not(lambda x: x > 0)
[-1, -2]
```

Parameters **func** – function to filter_not on

Returns filtered sequence

:rtype FunctionalSequence

find (*func*)
Finds the first element of the sequence that satisfies f. If no such element exists, then return None.

```
>>> seq(["abc", "ab", "bc"]).find(lambda x: len(x) == 2)
'ab'
```

Parameters **func** – function to find with

Returns first element to satisfy f or None

first ()
Returns the first element of the sequence.

```
>>> seq([1, 2, 3]).first()
1
```

Raises IndexError when the sequence is empty.

```
>>> seq([]).first()
Traceback (most recent call last):
  ...
IndexError: list index out of range
```

Returns first element of sequence

flat_map (*func*)
Applies f to each element of the sequence, which themselves should be sequences. Then appends each element of each sequence to a final result

```
>>> seq([[1, 2], [3, 4], [5, 6]]).flat_map(lambda x: x)
[1, 2, 3, 4, 5, 6]
```

```
>>> seq(["a", "bc", "def"]).flat_map(list)
['a', 'b', 'c', 'd', 'e', 'f']
```

```
>>> seq([[1], [2], [3]]).flat_map(lambda x: x * 2)
[1, 1, 2, 2, 3, 3]
```

Parameters **func** – function to apply to each sequence in the sequence

Returns application of f to elements followed by flattening

:rtype FunctionalSequence

flatten()
Flattens a sequence of sequences to a single sequence of elements.

```
>>> seq([[1, 2], [3, 4], [5, 6]])
[1, 2, 3, 4, 5, 6]
```

Returns flattened sequence

:rtype FunctionalSequence

fold_left(*zero_value*, *func*)
Takes a zero_value and performs a reduction to a value using f. f should take two parameters, 1) the value to accumulate into the result and 2) the current result. Elements are folded left to right.

```
>>> seq('a', 'bc', 'de', 'f', 'm', 'nop').fold_left("Start:", lambda v, curr: curr + 2 * v)
'Start:aabcbcdedeffmmnopnop'
```

Parameters

- **zero_value** – zero value to reduce into
- **func** – Two parameter function. First parameter is value to be accumulated into result. Second parameter is the current result

Returns value from folding values with f into zero_value

fold_right(*zero_value*, *func*)
Takes a zero_value and performs a reduction to a value using f. f should take two parameters, 1) the value to accumulate into the result and 2) the current result. Elements are folded right to left.

```
>>> seq('a', 'bc', 'de', 'f', 'm', 'nop').fold_right("Start:", lambda v, curr: curr + 2 * v)
'Start:nopnopmmffdedebcbcaa'
```

Parameters

- **zero_value** – zero value to reduce into
- **func** – Two parameter function. First parameter is value to be accumulated into result. Second parameter is the current result

Returns value from folding values with f into zero_value

:rtype FunctionalSequence

for_all(*func*)
Returns True if all elements in sequence make f evaluate to True.

```
>>> seq([1, 2, 3]).for_all(lambda x: x > 0)
True
```

```
>>> seq([1, 2, -1]).for_all(lambda x: x > 0)
False
```

Parameters **func** – function to check truth value of all elements with

Returns True if all elements make f evaluate to True

for_each(*func*)

Executes f on each element of the sequence.

```
>>> l = []
>>> seq([1, 2, 3, 4]).for_each(l.append)
>>> l
[1, 2, 3, 4]
```

Parameters **func** – function to execute

Returns None

group_by(*func*)

Group elements into a list of (Key, Value) tuples where f creates the key and maps to values matching that key.

```
>>> seq(["abc", "ab", "z", "f", "qw"]).group_by(len)
[(1, ['z', 'f']), (2, ['ab', 'qw']), (3, ['abc'])]
```

Parameters **func** – group by result of this function

Returns grouped sequence

:rtype FunctionalSequence

group_by_key()

Group sequence of (Key, Value) elements by Key.

```
>>> seq([('a', 1), ('b', 2), ('b', 3), ('b', 4), ('c', 3), ('c', 0)]).group_by_key()
[('a', [1]), ('c', [3, 0]), ('b', [2, 3, 4])]
```

Returns sequence grouped by key

:rtype FunctionalSequence

grouped(*size*)

Partitions the elements into groups of length size.

```
>>> seq([1, 2, 3, 4, 5, 6, 7, 8]).grouped(2)
[[1, 2], [3, 4], [5, 6], [7, 8]]
```

```
>>> seq([1, 2, 3, 4, 5, 6, 7, 8]).grouped(3)
[[1, 2, 3], [4, 5, 6], [7, 8]]
```

The last partition will be at least of size 1 and no more than length size :param size: size of the partitions :return: sequence partitioned into groups of length size :rtype FunctionalSequence

head()

Returns the first element of the sequence.

```
>>> seq([1, 2, 3]).head()
1
```

Raises IndexError when the sequence is empty.

```
>>> seq([]).head()
Traceback (most recent call last):
 ...
IndexError: list index out of range
```

Returns first element of sequence

head_option()
Returns the first element of the sequence or None, if the sequence is empty.

```
>>> seq([1, 2, 3]).head_option()
1
```

```
>>> seq([]).head_option()
None
```

Returns first element of sequence or None if sequence is empty

init()
Returns the sequence, without its last element.

```
>>> seq([1, 2, 3]).init()
[1, 2]
```

Returns sequence without last element

:rtype FunctionalSequence

inits()
Returns consecutive inits of the sequence.

```
>>> seq([1, 2, 3]).inits()
[[1, 2, 3], [1, 2], [1], []]
```

Returns consecutive init()s on sequence

:rtype FunctionalSequence

inner_join(*other*)
Sequence and other must be composed of (Key, Value) pairs. If self.sequence contains (K, V) pairs and other contains (K, W) pairs, the return result is a sequence of (K, (V, W)) pairs. Will return only elements where the key exists in both sequences.

```
>>> seq([('a', 1), ('b', 2), ('c', 3)]).inner_join([('a', 2), ('c', 5)])
[('a', (1, 2)), ('c', (3, 5))]
```

Parameters **other** – sequence to join with

Returns joined sequence of (K, (V, W)) pairs

:rtype FunctionalSequence

intersection(*other*)
New sequence with unique elements present in sequence and other.

```
>>> seq([1, 1, 2, 3]).intersection([2, 3, 4])
[2, 3]
```

Parameters **other** – sequence to perform intersection with

Returns intersection of sequence and other

:rtype FunctionalSequence

join(*other*, *join_type='inner'*)
Sequence and other must be composed of (Key, Value) pairs. If self.sequence contains (K, V) pairs and other contains (K, W) pairs, the return result is a sequence of (K, (V, W)) pairs. If join_type is "left", V values will always be present, W values may be present or None. If join_type is "right", W values will always be present, W values may be present or None. If join_type is "outer", V or W may be present or None, but never at the same time.

```
>>> seq([('a', 1), ('b', 2), ('c', 3)]).join([('a', 2), ('c', 5)], "inner")
[('a', (1, 2)), ('c', (3, 5))]
```

```
>>> seq([('a', 1), ('b', 2), ('c', 3)]).join([('a', 2), ('c', 5)])
[('a', (1, 2)), ('c', (3, 5))]
```

```
>>> seq([('a', 1), ('b', 2)]).join([('a', 3), ('c', 4)], "left")
[('a', (1, 3)), ('b', (2, None)]
```

```
>>> seq([('a', 1), ('b', 2)]).join([('a', 3), ('c', 4)], "right")
[('a', (1, 3)), ('c', (None, 4)]
```

```
>>> seq([('a', 1), ('b', 2)]).join([('a', 3), ('c', 4)], "outer")
[('a', (1, 3)), ('b', (2, None)), ('c', (None, 4))]
```

Parameters

- **other** – sequence to join with
- **join_type** – specifies join_type, may be "left", "right", or "outer"

Returns side joined sequence of (K, (V, W)) pairs

:rtype FunctionalSequence

last()
Returns the last element of the sequence.

```
>>> seq([1, 2, 3]).last()
3
```

Raises IndexError when the sequence is empty.

```
>>> seq([]).last()
Traceback (most recent call last):
  ...
IndexError: list index out of range
```

Returns last element of sequence

last_option()
Returns the last element of the sequence or None, if the sequence is empty.

```
>>> seq([1, 2, 3]).last_option()
3
```

```
>>> seq([]).last_option()
None
```

Returns last element of sequence or None if sequence is empty

left_join(*other*)
Sequence and other must be composed of (Key, Value) pairs. If self.sequence contains (K, V) pairs and other contains (K, W) pairs, the return result is a sequence of (K, (V, W)) pairs. V values will always be present, W values may be present or None.

```
>>> seq([('a', 1), ('b', 2)]).join([('a', 3), ('c', 4)])
[('a', (1, 3)), ('b', (2, None)]
```

Parameters **other** – sequence to join with

Returns left joined sequence of (K, (V, W)) pairs

:rtype FunctionalSequence

len()
Return length of sequence using its length function.

```
>>> seq([1, 2, 3]).len()
3
```

Returns length of sequence

list()
Converts sequence to list of elements.

```
>>> type(seq([]).list())
list
```

```
>>> type(seq([]))
functional.chain.FunctionalSequence
```

```
>>> seq([1, 2, 3]).list()
[1, 2, 3]
```

Returns list of elements in sequence

make_string(*separator*)
Concatenate the elements of the sequence into a string separated by separator.

```
>>> seq([1, 2, 3]).make_string("@")
'1@2@3'
```

Parameters **separator** – string separating elements in string

Returns concatenated string separated by separator

map (*f*)
Maps f onto the elements of the sequence.

```
>>> seq([1, 2, 3, 4]).map(lambda x: x * -1)
[-1, -2, -3, -4]
```

Parameters **f** – function to map with

Returns sequence with f mapped onto it

:rtype FunctionalSequence

max ()
Returns the largest element in the sequence. If the sequence has multiple maximal elements, only the first one is returned.

The compared objects must have defined comparison methods. Raises TypeError when the objects are not comparable.

The sequence can not be empty. Raises ValueError when the sequence is empty.

```
>>> seq([2, 4, 5, 1, 3]).max()
5
```

```
>>> seq('aa', 'xyz', 'abcd', 'xyy').max()
'xyz'
```

```
>>> seq([1, "a"]).max()
Traceback (most recent call last):
 ...
TypeError: unorderable types: int() < str()
```

```
>>> seq([]).max()
Traceback (most recent call last):
 ...
ValueError: max() arg is an empty sequence
```

max_by (*func*)
Returns the largest element in the sequence. Provided function is used to generate key used to compare the elements. If the sequence has multiple maximal elements, only the first one is returned.

The sequence can not be empty. Raises ValueError when the sequence is empty.

```
>>> seq([2, 4, 5, 1, 3]).max_by(lambda num: num % 4)
3
```

```
>>> seq('aa', 'xyz', 'abcd', 'xyy').max_by(len)
'abcd'
```

```
>>> seq([]).max_by(lambda x: x)
Traceback (most recent call last):
 ...
ValueError: max() arg is an empty sequence
```

min ()
Returns the smallest element in the sequence. If the sequence has multiple minimal elements, only the first one is returned.

The compared objects must have defined comparison methods. Raises TypeError when the objects are not comparable.

The sequence can not be empty. Raises ValueError when the sequence is empty.

```
>>> seq([2, 4, 5, 1, 3]).min()
1
```

```
>>> seq('aa', 'xyz', 'abcd', 'xyy').min()
'aa'
```

```
>>> seq([1, "a"]).min()
Traceback (most recent call last):
 ...
TypeError: unorderable types: int() < str()
```

```
>>> seq([]).min()
Traceback (most recent call last):
 ...
ValueError: min() arg is an empty sequence
```

min_by (*func*)

Returns the smallest element in the sequence. Provided function is used to generate key used to compare the elements. If the sequence has multiple minimal elements, only the first one is returned.

The sequence can not be empty. Raises ValueError when the sequence is empty.

```
>>> seq([2, 4, 5, 1, 3]).min_by(lambda num: num % 6)
5
```

```
>>> seq('aa', 'xyz', 'abcd', 'xyy').min_by(len)
'aa'
```

```
>>> seq([]).min_by(lambda x: x)
Traceback (most recent call last):
 ...
ValueError: min() arg is an empty sequence
```

non_empty()

Returns True if the sequence does not have length zero.

```
>>> seq([]).non_empty()
False
```

```
>>> seq([1]).non_empty()
True
```

Returns True if sequence length is not zero

outer_join (*other*)

Sequence and other must be composed of (Key, Value) pairs. If self.sequence contains (K, V) pairs and other contains (K, W) pairs, the return result is a sequence of (K, (V, W)) pairs. One of V or W will always be not None, but the other may be None

```
>>> seq([('a', 1), ('b', 2)]).outer_join([('a', 3), ('c', 4)], "outer")
[('a', (1, 3)), ('b', (2, None)), ('c', (None, 4))]
```

Parameters **other** – sequence to join with

Returns outer joined sequence of (K, (V, W)) pairs

:rtype FunctionalSequence

partition (*func*)
Partition the sequence based on satisfying the predicate f.

```
>>> seq([-1, 1, -2, 2]).partition(lambda x: x < 0)
([-1, -2], [1, 2])
```

Parameters **func** – predicate to partition on

Returns tuple of partitioned sequences

:rtype FunctionalSequence

product ()
Takes product of elements in sequence.

```
>>> seq([1, 2, 3, 4]).product()
24
```

```
>>> seq([]).product()
1
```

Returns product of elements in sequence

reduce (*func*)
Reduce sequence of elements using f.

```
>>> seq([1, 2, 3]).reduce(lambda x, y: x + y)
6
```

Parameters **func** – two parameter, associative reduce function

Returns reduced value using f

:rtype FunctionalSequence

reduce_by_key (*func*)
Reduces a sequence of (Key, Value) using f on each Key.

```
>>> seq([('a', 1), ('b', 2), ('b', 3), ('b', 4), ('c', 3), ('c', 0)])          .reduc
[('a', 1), ('c', 3), ('b', 9)]
```

Parameters **func** – reduce each list of values using two parameter, associative f

Returns Sequence of tuples where the value is reduced with f

:rtype FunctionalSequence

reverse ()
Returns the reversed sequence.

```
>>> seq([1, 2, 3]).reverse()
[3, 2, 1]
```

Returns reversed sequence

:rtype FunctionalSequence

right_join(*other*)
Sequence and other must be composed of (Key, Value) pairs. If self.sequence contains (K, V) pairs and other contains (K, W) pairs, the return result is a sequence of (K, (V, W)) pairs. W values will always bepresent, V values may be present or None.

```
>>> seq([('a', 1), ('b', 2)]).join([('a', 3), ('c', 4)])
[('a', (1, 3)), ('b', (2, None)]
```

Parameters **other** – sequence to join with

Returns right joined sequence of (K, (V, W)) pairs

:rtype FunctionalSequence

sequence

set()
Converts sequence to a set of elements.

```
>>> type(seq([])).to_set()
set
```

```
>>> type(seq([]))
functional.chain.FunctionalSequence
```

```
>>> seq([1, 1, 2, 2]).set()
{1, 2}
```

:return:set of elements in sequence

size()
Return size of sequence using its length function.

Returns size of sequence

slice(*start*, *until*)
Takes a slice of the sequence starting at start and until but not including until.

```
>>> seq([1, 2, 3, 4]).slice(1, 2)
[2]
>>> seq([1, 2, 3, 4]).slice(1, 3)
[2, 3]
```

Parameters

- **start** – starting index
- **until** – ending index

Returns slice including start until but not including until

:rtype FunctionalSequence

sorted(*key=None*, *reverse=False*)
Uses python sort and its passed arguments to sort the input.

```
>>> seq([2, 1, 4, 3]).sorted()
[1, 2, 3, 4]
```

Parameters

- **key** –
- **reverse** – return list reversed or not

Returns sorted sequence

:rtype FunctionalSequence

sum()
Takes sum of elements in sequence.

```
>>> seq([1, 2, 3, 4]).sum()
10
```

Returns sum of elements in sequence

symmetric_difference(*other*)
New sequence with elements in either sequence or other, but not both.

```
>>> seq([1, 2, 3, 3]).symmetric_difference([2, 4, 5])
[1, 3, 4, 5]
```

Parameters **other** – sequence to perform symmetric difference with

Returns symmetric difference of sequence and other

:rtype FunctionalSequence

tail()
Returns the sequence, without its first element.

```
>>> seq([1, 2, 3]).init()
[2, 3]
```

Returns sequence without first element

:rtype FunctionalSequence

tails()
Returns consecutive tails of the sequence.

```
>>> seq([1, 2, 3]).tails()
[[1, 2, 3], [2, 3], [3], []]
```

Returns consecutive tail()s of the sequence

:rtype FunctionalSequence

take(*n*)
Take the first n elements of the sequence.

```
>>> seq([1, 2, 3, 4]).take(2)
[1, 2]
```

Parameters **n** – number of elements to take

Returns first n elements of sequence

:rtype FunctionalSequence

take_while(*func*)
Take elements in the sequence until f evaluates to False, then return them.

```
>>> seq([1, 2, 3, 4, 5, 1, 2]).take_while(lambda x: x < 3)
[1, 2]
```

Parameters **func** – truth returning function

Returns elements taken until f evaluates to False

:rtype FunctionalSequence

to_dict()
Converts sequence of (Key, Value) pairs to a dictionary.

```
>>> type(seq([('a', 1)]).to_dict())
dict
```

```
>>> type(seq([]))
functional.chain.FunctionalSequence
```

```
>>> seq([('a', 1), ('b', 2)]).to_dict()
{'a': 1, 'b': 2}
```

Returns dictionary from sequence of (Key, Value) elements

to_list()
Converts sequence to list of elements.

```
>>> type(seq([]).to_list())
list
```

```
>>> type(seq([]))
functional.chain.FunctionalSequence
```

```
>>> seq([1, 2, 3]).to_list()
[1, 2, 3]
```

Returns list of elements in sequence

to_set()
Converts sequence to a set of elements.

```
>>> type(seq([])).to_set()
set
```

```
>>> type(seq([]))
functional.chain.FunctionalSequence
```

```
>>> seq([1, 1, 2, 2]).to_set()
{1, 2}
```

:return:set of elements in sequence

union(*other*)
New sequence with unique elements from self.sequence and other.

```
>>> seq([1, 1, 2, 3, 3]).union([1, 4, 5])
[1, 2, 3, 4, 5]
```

Parameters **other** – sequence to union with

Returns union of sequence and other

:rtype FunctionalSequence

zip(*sequence*)
Zips the stored sequence with the given sequence.

```
>>> seq([1, 2, 3]).zip([4, 5, 6])
[(1, 4), (2, 5), (3, 6)]
```

Parameters **sequence** – second sequence to zip

Returns stored sequence zipped with given sequence

:rtype FunctionalSequence

zip_with_index()
Zips the sequence to its index, with the index being the first element of each tuple.

```
>>> seq(['a', 'b', 'c']).zip_with_index()
[(0, 'a'), (1, 'b'), (2, 'c')]
```

Returns sequence zipped to its index

:rtype FunctionalSequence

functional.chain.**seq**(**args*)
Alias function for creating a new FunctionalSequence. Additionally it also parses various types of input to a FunctionalSequence as best it can.

```
>>> type(seq([1, 2]))
functional.chain.FunctionalSequence
```

```
>>> type(FunctionalSequence([1, 2]))
functional.chain.FunctionalSequence
```

```
>>> seq([1, 2, 3])
[1, 2, 3]
```

```
>>> seq(1, 2, 3)
[1, 2, 3]
```

```
>>> seq(1)
[1]
```

```
>>> seq(range(4))
[0, 1, 2, 3]
```

Parameters **args** – Three types of arguments are valid. 1) Iterable which is then directly wrapped as a FunctionalSequence 2) A list of arguments is converted to a FunctionalSequence 3) A single non-iterable is converted to a single element FunctionalSequence

:rtype FunctionalSequence :return: wrapped sequence

1.3 Module contents

Package which supplies utilities primarily through functional.seq to do pipeline style, functional transformations and reductions.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

f