fntools Documentation

Release 1.0

Taurus Olson

March 04, 2016

Contents

Functional programming tools for data processing

fntools is a simple library providing the user with functional programming functions to transform, filter and inspect Python data structures.

CHAPTER 1

API

1.1 Transformation

fntools.**zip_with**(*fn*, **colls*)

Return the result of the function applied on the zip of the collections

Parameters

- **fn** – a function
- **colls** – collections

Returns an iterator

```
>>> list(zip_with(lambda x, y: x-y, [10, 20, 30], [42, 19, 43]))
[-32, 1, -13]
```

fntools.**concat**(*colls*)

Concatenate a list of collections

Parameters **colls** – a list of collections

Returns the concatenation of the collections

```
>>> concat(([1, 2], [3, 4]))
[1, 2, 3, 4]
```

fntools.**mapcat**(*fn*, *colls*)

Concatenate the result of a map

Parameters

- **fn** – a function
- **colls** – a list of collections

Returns a list

```
>>> mapcat(reversed,  [[3, 2, 1, 0], [6, 5, 4], [9, 8, 7]])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

fntools.**dmap**(*fn*, *record*)
map for a directory

Parameters

- **fn** – a function
- **record** – a dictionary

Returns a dictionary

```
>>> grades = [{'math': 13, 'biology': 17, 'chemistry': 18},
... {'math': 15, 'biology': 12, 'chemistry': 13},
... {'math': 16, 'biology': 17, 'chemistry': 11}]
```

```
>>> def is_greater_than(x):
...     def func(y):
...         return y > x
...     return func
```

```
>>> dmap(is_greater_than(15), grades[0])
{'biology': True, 'chemistry': True, 'math': False}
```

fntools.**compose**(**fns*)
Return the function composed with the given functions

Parameters **fns** – functions

Returns a function

```
>>> add2 = lambda x: x+2
>>> mult3 = lambda x: x*3
>>> new_fn = compose(add2, mult3)
>>> new_fn(2)
8
```

Note: compose(fn1, fn2, fn3) is the same as fn1(fn2(fn3)) which means that the last function provided is the first to be applied.

---

fntools.**groupby**(*fn*, *coll*)
Group elements in sub-collections by fn

Parameters

- **fn** – a function
- **coll** – a collection

Returns a dictionary

```
>>> groupby(len, ['John', 'Terry', 'Eric', 'Graham', 'Mickael'])
{4: ['John', 'Eric'], 5: ['Terry'], 6: ['Graham'], 7: ['Mickael']}
```

fntools.**reductions**(*fn*, *seq*, *acc=None*)
Return the intermediate values of a reduction

Parameters

- **fn** – a function
- **seq** – a sequence
- **acc** – the accumulator

Returns a list

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

```
>>> reductions(lambda x, y: x + y, [1, 2, 3], 10)
[11, 13, 16]
```

fntools.**split**(*coll*, *factor*)

Split a collection by using a factor

Parameters

- **coll** – a collection
- **factor** – a collection of factors

Returns a dictionary

```
>>> bands = ('Led Zeppelin', 'Debussy', 'Metallica', 'Iron Maiden', 'Bach')
>>> styles = ('rock', 'classic', 'rock', 'rock', 'classic')
>>> split(bands, styles)
{'classic': ['Debussy', 'Bach'], 'rock': ['Led Zeppelin', 'Metallica', 'Iron Maiden']}
```

fntools.**assoc**(*_d*, *key*, *value*)

Associate a key with a value in a dictionary

Parameters

- **_d** – a dictionary
- **key** – a key in the dictionary
- **value** – a value for the key

Returns a new dictionary

```
>>> data = {}
>>> new_data = assoc(data, 'name', 'Holy Grail')
>>> new_data
{'name': 'Holy Grail'}
>>> data
{}
```

Note: the original dictionary is not modified

fntools.**dispatch**(*data*, *fns*)

Apply the functions on the data

Parameters

- **data** – the data
- **fns** – a list of functions

Returns a collection

```
>>> x = (1, 42, 5, 79)
>>> dispatch(x, (min, max))
[1, 79]
```

fntools.**multimap**(*fn*, *colls*)

Apply a function on multiple collections

Parameters

- **fn** – a function
- **colls** – collections

Returns a collection

```
>>> multimap(operator.add, ((1, 2, 3), (4, 5, 6)))
[5, 7, 9]
```

```
>>> f = lambda x, y, z: 2*x + 3*y - z
>>> result = multimap(f, ((1, 2), (4, 1), (1, 1)))
>>> result[0] == f(1, 4, 1)
True
>>> result[1] == f(2, 1, 1)
True
```

1.2 Filtering

fntools.**duplicates**(*coll*)

Return the duplicated items in the given collection

Parameters **coll** – a collection

Returns a list of the duplicated items in the collection

```
>>> duplicates([1, 1, 2, 3, 3, 4, 1, 1])
[1, 3]
```

fntools.**pluck**(*record*, **keys*, ***kwargs*)

Return the record with the selected keys

Parameters

- **record** – a list of dictionaries
- **keys** – some keys from the record
- **kwargs** – keywords determining how to deal with the keys

```
>>> d = {'name': 'Lancelot', 'actor': 'John Cleese', 'color': 'blue'}
>>> pluck(d, 'name', 'color')
{'color': 'blue', 'name': 'Lancelot'}
```

the keyword 'default' allows to replace a None value >>> d = {'year': 2014, 'movie': 'Bilbo'} >>> pluck(d, 'year', 'movie', 'nb_aliens', default=0) {'movie': 'Bilbo', 'nb_aliens': 0, 'year': 2014}

fntools.**get_in**(*record*, **keys*, ***kwargs*)

Return the value corresponding to the keys in a nested record

Parameters

- **record** – a dictionary
- **keys** – strings
- **kwargs** – keywords

Returns the value for the keys

```
>>> d = {'id': {'name': 'Lancelot', 'actor': 'John Cleese', 'color': 'blue'}}
>>> get_in(d, 'id', 'name')
'Lancelot'
```

```
>>> get_in(d, 'id', 'age', default='?')
'?'
```

fntools.**valueof**(*records*, *key*)

Extract the value corresponding to the given key in all the dictionaries

```
>>> bands = [{'name': 'Led Zeppelin', 'singer': 'Robert Plant', 'guitarist': 'Jimmy Page'},
... {'name': 'Metallica', 'singer': 'James Hetfield', 'guitarist': 'Kirk Hammet'}]
>>> valueof(bands, 'singer')
['Robert Plant', 'James Hetfield']
```

fntools.**dfilter**(*fn*, *record*)

filter for a directory

Parameters

- **fn** – A predicate function
- **record** – a dict

Returns a dict

```
>>> odd = lambda x: x % 2 != 0
>>> dfilter(odd, {'Terry': 30, 'Graham': 35, 'John': 27})
{'John': 27, 'Graham': 35}
```

fntools.**find**(*fn*, *record*)

Apply a function on the record and return the corresponding new record

Parameters

- **fn** – a function
- **record** – a dictionary

Returns a dictionary

```
>>> find(max, {'Terry': 30, 'Graham': 35, 'John': 27})
{'Graham': 35}
```

1.3 Inspection

fntools.**isiterable**(*coll*)

Return True if the collection is any iterable except a string

Parameters **coll** – a collection

Returns a boolean

```
>>> isiterable(1)
False
>>> isiterable('iterable')
False
>>> isiterable([1, 2, 3])
True
```

fntools.**are_in**(*items*, *collection*)
Return True for each item in the collection

Parameters

- **items** – a sub-collection
- **collection** – a collection

Returns a list of booleans

```
>>> are_in(['Terry', 'James'], ['Terry', 'John', 'Eric'])
[True, False]
```

fntools.**any_in**(*items*, *collection*)
Return True if any of the items are in the collection

Parameters

- **items** – items that may be in the collection
- **collection** – a collection

Returns a boolean

```
>>> any_in(2, [1, 3, 2])
True
>>> any_in([1, 2], [1, 3, 2])
True
>>> any_in([1, 2], [1, 3])
True
```

fntools.**all_in**(*items*, *collection*)
Return True if all of the items are in the collection

Parameters

- **items** – items that may be in the collection
- **collection** – a collection

Returns a boolean

```
>>> all_in(2, [1, 3, 2])
True
>>> all_in([1, 2], [1, 3, 2])
True
>>> all_in([1, 2], [1, 3])
False
```

fntools.**occurrences**(*coll*, *value=None*, ***options*)
Return the occurrences of the elements in the collection

Parameters

- **coll** – a collection
- **value** – a value in the collection
- **options** – an optional keyword used as a criterion to filter the

values in the collection :returns: the frequency of the values in the collection as a dictionary

```
    >>> occurrences((1, 1, 2, 3))
    {1: 2, 2: 1, 3: 1}
    >>> occurrences((1, 1, 2, 3), 1)
    2
```

Filter the values of the occurrences that # are <, <=, >, >=, == or != than a given number >>> occurrences((1, 1, 2, 3), lt=3) {1: 2, 2: 1, 3: 1} >>> occurrences((1, 1, 2, 3), gt=1) {1: 2} >>> occurrences((1, 1, 2, 3), ne=1) {1: 2}

fntools.**indexof**(*coll*, *item*, *start=0*, *default=None*)

Return the index of the item in the collection

Parameters

- **coll** – iterable
- **item** – scalar
- **start** – (optional) The start index

Default The default value of the index if the item is not in the collection

Returns idx – The index of the item in the collection

```
    >>> monties = ['Eric', 'John', 'Terry', 'Terry', 'Graham', 'Mickael']
    >>> indexof(monties, 'Terry')
    2
```

```
    >>> indexof(monties, 'Terry', start=3)
    3
```

```
    >>> indexof(monties, 'Terry', start=4) is None
    True
```

fntools.**indexesof**(*coll*, *item*)

Return all the indexes of the item in the collection

Parameters

- **coll** – the collection
- **item** – a value

Returns a list of indexes

```
    >>> monties = ['Eric', 'John', 'Terry', 'Terry', 'Graham', 'Mickael']
    >>> indexesof(monties, 'Terry')
    [2, 3]
```

fntools.**monotony**(*seq*)

Determine the monotony of a sequence

Parameters **seq** – a sequence

Returns 1 if the sequence is sorted (increasing)

Returns 0 if it is not sorted

Returns -1 if it is sorted in reverse order (decreasing)

```
    >>> monotony([1, 2, 3])
    1
    >>> monotony([1, 3, 2])
    0
```

```
>>> monotony([3, 2, 1])
-1
```

fntools.**attributes**(*data*)

Return all the non callable and non special attributes of the input data

Parameters **data** – an object

Returns a list

```
>>> class table:
...     def __init__(self, name, rows, cols):
...         self.name = name
...         self.rows = rows
...         self.cols = cols
```

```
>>> t = table('people', 100, 3)
>>> attributes(t)
['cols', 'name', 'rows']
```

fntools.**count**(*fn*, *coll*)

Return the count of True values returned by the predicate function applied to the collection

Parameters

- **fn** – a predicate function
- **coll** – a collection

Returns an integer

```
>>> count(lambda x: x % 2 == 0, [11, 22, 31, 24, 15])
2
```

fntools.**isdistinct**(*coll*)

Return True if all the items in the collections are distinct.

Parameters **coll** – a collection

Returns a boolean

```
>>> isdistinct([1, 2, 3])
True
>>> isdistinct([1, 2, 2])
False
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

f