fntools Documentation

Release 1.1.2

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. It introduces no new class or data structures but instead emphasizes the use of:

- pure
- composable
- lightweight

functions to solve common problems while processing your data.

CHAPTER 1

API

1.1 Transformation

fntools.**use_with**(*data*, *fn*, **attrs*)

Apply a function on the attributes of the data

Parameters

- **data** – an object
- **fn** – a function
- **attrs** – some attributes of the object

Returns an object

Let's create some data first:

```
>>> from collections import namedtuple
>>> Person = namedtuple('Person', ('name', 'age', 'gender'))
>>> alice = Person('Alice', 30, 'F')
```

Usage:

```
>>> make_csv_row = lambda n, a, g: '%s,%d,%s' % (n, a, g)
>>> use_with(alice, make_csv_row, 'name', 'age', 'gender')
'Alice,30,F'
```

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.**unzip**(*colls*)
Unzip collections

Parameters **colls** – collections

Returns unzipped collections

```
>>> unzip([[1, 2, 3], [10, 20, 30], [100, 200, 300]])
[(1, 10, 100), (2, 20, 200), (3, 30, 300)]
```

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.**rmap**(*fn*, *coll*, *is_iterable=None*)
A recursive map

Parameters

- **fn** – a function
- **coll** – a list
- **isiterable** – a predicate function determining whether a value is iterable.

Returns a list

```
>>> rmap(lambda x: 2*x, [1, 2, [3, 4]])
[2, 4, [6, 8]]
```

fntools.**replace**(*x*, *old*, *new*, *fn=<built-in function eq>*)

Replace x with new if fn(x, old) is True.

Parameters

- **x** – Any value
- **old** – The old value we want to replace
- **new** – The value replacing old
- **fn** – The predicate function determining the relation between x and old. By default fn is the equality function.

Returns x or new

```
>>> map(lambda x: replace(x, None, -1), [None, 1, 2, None])
[-1, 1, 2, -1]
```

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
```

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

Apply a function on multiple collections

Parameters

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

Returns a collection

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

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

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

Apply functions recursively on your data

Parameters

- **data** – the data
- **fns** – functions

Returns an object

```
    >>> inc = lambda x: x + 1
    >>> pipe(42, inc, str)
    '43'
```

fntools.**pipe_each**(*coll*, **fns*)

Apply functions recursively on your collection of data

Parameters

- **coll** – a collection
- **fns** – functions

Returns a list

```
>>> inc = lambda x: x + 1
>>> pipe_each([0, 1, 1, 2, 3, 5], inc, str)
['1', '2', '2', '3', '4', '6']
```

fntools.**shift**(*func*, **args*, ***kwargs*)

This function is basically a beefed up lambda x: func(x, *args, **kwargs)

shift() comes in handy when it is used in a pipeline with a function that needs the passed value as its first argument.

Parameters

- **func** – a function
- **args** – objects
- **kwargs** – keywords

```
>>> def div(x, y): return float(x) / y
```

This is equivalent to div(42, 2):

```
>>> shift(div, 2)(42)
21.0
```

which is different from div(2, 42):

```
>>> from functools import partial
>>> partial(div, 2)(42)
0.047619047619047616
```

fntools.**repeatedly**(*func*)

Repeat a function taking no argument

Parameters **func** – a function

Returns a generator

```
>>> import random as rd
>>> rd.seed(123)
>>> take(3, repeatedly(rd.random))
[0.052363598850944326, 0.08718667752263232, 0.4072417636703983]
```

fntools.**update**(*records*, *column*, *values*)

Update the column of records

Parameters

- **records** – a list of dictionaries
- **column** – a string
- **values** – an iterable or a function

Returns new records with the columns updated

```
>>> movies = [
... {'title': 'The Holy Grail', 'year': 1975, 'budget': 4E5, 'total_gross': 5E6},
... {'title': 'Life of Brian', 'year': 1979, 'budget': 4E6, 'total_gross': 20E6},
... {'title': 'The Meaning of Life', 'year': 1983, 'budget': 9E6, 'total_gross': 14.9E6}
... ]
>>> new_movies = update(movies, 'budget', lambda x: 2*x)
>>> [new_movies[i]['budget'] for i,_ in enumerate(movies)]
[800000.0, 8000000.0, 18000000.0]
```

```
>>> new_movies2 = update(movies, 'budget', (40, 400, 900))
>>> [new_movies2[i]['budget'] for i,_ in enumerate(movies)]
[40, 400, 900]
```

fntools.**use**(*data*, *attrs*)

Return the values of the attributes for the given data

Parameters

- **data** – the data
- **attrs** – strings

Returns a list

With a dict:

```
>>> band = {'name': 'Metallica', 'singer': 'James Hetfield', 'guitarist': 'Kirk Hammet'}
>>> use(band, ('name', 'date', 'singer'))
['Metallica', None, 'James Hetfield']
```

With a non dict data structure:

```
>>> from collections import namedtuple
>>> Person = namedtuple('Person', ('name', 'age', 'gender'))
>>> alice = Person('Alice', 30, 'F')
>>> use(alice, ('name', 'gender'))
['Alice', 'F']
```

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']
```

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.**pluck_each**(*records*, *columns*)

Return the records with the selected columns

Parameters

- **records** – a list of dictionaries
- **columns** – a list or a tuple

Returns a list of dictionaries with the selected columns

```
>>> movies = [
... {'title': 'The Holy Grail', 'year': 1975, 'budget': 4E5, 'total_gross': 5E6},
... {'title': 'Life of Brian', 'year': 1979, 'budget': 4E6, 'total_gross': 20E6},
... {'title': 'The Meaning of Life', 'year': 1983, 'budget': 9E6, 'total_gross': 14.9E6}
... ]
>>> pluck_each(movies, ('title', 'year'))
[{'year': 1975, 'title': 'The Holy Grail'}, {'year': 1979, 'title': 'Life of Brian'}, {'year': 1
```

fntools.**take**(*n*, *seq*)

Return the n first items in the sequence

Parameters

- **n** – an integer
- **seq** – a sequence

Returns a list

```
>>> take(3, xrange(10000))
[0, 1, 2]
```

fntools.**drop**(*n*, *seq*)
Return the n last items in the sequence

Parameters

- **n** – an integer
- **seq** – a sequence

Returns a list

```
>>> drop(9997, xrange(10000))
[9997, 9998, 9999]
```

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}
```

fntools.**find_each**(*fn*, *records*)
Apply a function on the records and return the corresponding new record

Parameters

- **fn** – a function
- **records** – a collection of dictionaries

Returns new records

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

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.**remove**(*coll*, *value*)
Remove all the occurrences of a given value

Parameters

- **coll** – a collection
- **value** – the value to remove

Returns a list

```
    >>> data = ('NA', 0, 1, 'NA', 1, 2, 3, 'NA', 5)
    >>> remove(data, 'NA')
    (0, 1, 1, 2, 3, 5)
```

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.**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.**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.**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.**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.**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
```

fntools.**nrow**(*records*)
Return the number of rows in the records

Parameters **records** – a list of dictionaries

Returns an integer

```
>>> movies = [
... {'title': 'The Holy Grail', 'year': 1975, 'budget': 4E5, 'total_gross': 5E6},
... {'title': 'Life of Brian', 'year': 1979, 'budget': 4E6, 'total_gross': 20E6},
... {'title': 'The Meaning of Life', 'year': 1983, 'budget': 9E6, 'total_gross': 14.9E6}
... ]
>>> nrow(movies)
3
```

fntools.**ncol**(*records*)
Return the number of columns in the records

Parameters **records** – a list of dictionaries

Returns an integer

```
>>> movies = [
... {'title': 'The Holy Grail', 'year': 1975, 'budget': 4E5, 'total_gross': 5E6},
... {'title': 'Life of Brian', 'year': 1979, 'budget': 4E6, 'total_gross': 20E6},
... {'title': 'The Meaning of Life', 'year': 1983, 'budget': 9E6, 'total_gross': 14.9E6}
... ]
>>> ncol(movies)
4
```

fntools.**names**(*records*)
Return the column names of the records

Parameters **records** – a list of dictionaries

Returns a list of strings

```
    >>> movies = [
    ... {'title': 'The Holy Grail', 'year': 1975, 'budget': 4E5, 'total_gross': 5E6},
    ... {'title': 'Life of Brian', 'year': 1979, 'budget': 4E6, 'total_gross': 20E6},
    ... {'title': 'The Meaning of Life', 'year': 1983, 'budget': 9E6, 'total_gross': 14.9E6}
    ... ]
    >>> names(movies)
    ['total_gross', 'year', 'budget', 'title']
```

CHAPTER 2

CHANGELOG

2.1 v1.1.2 / 2016-03-04

- Bump to version 1.1.2
- Add Makefile
- Update documentation
- Minor fix
- Update documentation
- Fix issue in __init__.py
- Fix issue in __init__.py
- Remove valuesof function and include it in use
- Fix compose
- Fix indexof
- Rename function: select -> pluck_each
- Fix mapcat
- Fix find_each
- Update __init__
- Ignore .projections.json for Vim
- Add new functions for inspecting data
- Remove useless whitespaces
- Add new functions

2.2 v1.1.1 / 2016-03-01

- Fix rmap

2.3 v1.1.0 / 2016-02-23

- Bump to version 1.1.0
- Update import in __init__.py
- Minor fix in setup.py and README.rst
- Add 2 new functions
- Implement a new function
- Implement 2 new functions
- Minor fixes.
- Minor fix
- Enhance occurrences
- Minor fix
- Add some informations into __init__.py
- Add more metadata for the package on PyPi
- Add MIT license and MANIFEST
- Add examples to the README.
- Ignore some files related to the packaging

2.4 v1.0.4 / 2014-12-06

- Bump version to 1.0.4
- Add functions into __init__.py

2.5 v1.0.3 / 2014-12-06

- Bump version to 1.0.3
- Fix setup.py

2.6 v1.0.2 / 2014-12-06

- Bump version to 1.0.2
- Minor fix
- Minor fix in README.md
- Change README.md to README.rst
- Add __init__.py

2.7 v1.0.1 / 2014-12-06

- Bump version to 1.0.1
- Add history
- Minor fixes
- Simplify setup.py
- Ignore experimental.py
- Minor changes
- Change path to module
- fntools is on pypi
- Improve setup.py
- Introduce new pipeline functions
- Rename issorted -> monotony
- [minor changes] Minor changes in README
- [minor change] Add readthedocs badge in README.md
- [minor change] Respect PEP8
- [fix doc] Remove files in _build
- [doc] Remove quickstart from the index
- [doc] Use default theme for doc
- [new] Add documentation
- [new file] Add .gitignore
- [minor change] Update README.md
- [new] Implement new functions
- [new functions] Implement new predicate functions
- [enhancement] pluck accepts keyword arguments for default value
- [new functions] Implement functions for inspecting data
- [new function] Implement dispatch
- Initial commit

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

f