Pyfunctools

Release 0.5.1

Natan Santos

Feb 23, 2023

CONTENTS

CHAPTER

ONE

INSTALLATION

Via PIP (recommended):

```
pip install pyfunctools
```

Via GitHub:

```
git clone https://github.com/natanfeitosa/pyfunctools.git && cd pyfunctools && pip␣
↪install .
```

Or the command line:

```
easy_install pyfunctools
```

CHAPTER

TWO

PYFUNCTOOLS PACKAGE

2.1 Pyfunctools

Pyfunctools is a module that provides functions, methods and classes that help in the creation of projects in python, bringing functional and object-oriented programming methods.

class pyfunctools.**Array**(**args*, *default=None*)

Bases: object

Class that has common methods in arrays that are not present in the builtin list class.

Raises **NotImplementedError** – Error thrown when a None value is passed in the constructor

Examples

```
>>> #Create an array/list
>>> Array()
<Array values=[] >
>>> Array(4)
<Array values=[None, None, None, None] >
>>> Array(1, 2)
<Array values=[1, 2] >
```

append(*x: any*)

Append a new item with value x to the end of the Array.

Parameters **x** (*any*) – object to add

Examples

```
>>> array = Array(1, 2, 3)
>>> array.append(4)
>>> array.to_list()
[1, 2, 3, 4]
```

chunk(*size: int = 1*) → List[list]

Divides the Array object into sublists.

Parameters **size** (*int, optional*) – Chunk size. Defaults to 1

Returns A new list containing the chunks of the Array object

Return type list

Examples

```
>>> array = Aray([1, 2, 3, 4])
>>> array.chunk()
[[1], [2], [3], [4]]
>>> array.chunk(2)
[[1, 2], [3, 4]]
>>> array.chunk(3)
[[1, 2, 3], [4, 5, 6]]
```

clone()
Clone this array instance

Returns A new array

Return type *Array*

concat(**args*)
Concatenate into Array

Returns The Array instance being changed

Return type *Array*

Examples

```
>>> a = Array([1, 2])
>>> a.concat(3)
<Array values=[1, 2, 3] >
>>> a.concat([4, 5], [6, 7])
<Array values=[1, 2, 3, 4, 5, 6, 7] >
>>> #Turning into list type
>>> a.to_list()
[1, 2, 3, 4, 5, 6, 7]
```

count(*x: any*) → int
Return the number of times x appears in the array.

Examples

```
>>> Array(1, 2, 3).count(1)
1
>>> Array(2, 0, 2, 2).count(2)
3
>>> Array(2, 0, 0, 2).count(1)
0
```

fill(*fill_with: any*)
Replaces/fill the values in the array without changing the len

Returns The Array instance being changed

Return type *Array*

Examples

```
>>> a = Array([1, 2])
>>> a.fill('a')
<Array values=['a', 'a'] >
>>> #Turning into list type
>>> a.to_list()
['a', 'a']
```

filter(*func*) → list

Method to filter a Aray item

Parameters **func** (*function*) – The callback function takes an item and index, and must return a boolean

Examples

```
>>> array = Array(1, 2, 3, 4)
>>> array.filter(lambda item, index: item % 2 == 0)
[2, 4]
```

forEach(*func*)

Calls a function for each item, passing the item itself and the index

Parameters **func** (*function*) – Callback function

Examples

```
>>> array = Array(1, 2)
>>> array.forEach(lambda item, index: print(f'{item}, {index}'))
1, 0
2, 1
```

includes(*item: any*) → bool

Test if an item exists in this Array

Parameters **item** (*any*) – Item to test

Examples

```
>>> array = Array(1, 2, 3)
>>> array.includes(4)
False
>>> array.includes(1)
True
```

index_of(*obj: any*) → int

The method returns the first index at which a given element can be found in the array, or -1 if it is not present.

Parameters **obj** (*any*) – Element to locate in the array.

Returns -1 if not exists in the array

Return type int

Examples

```
>>> array = Array(1, 2, 3)
>>> array.index_of(4)
-1
>>> array.index_of(1)
0
```

static is_array(*arr: any*) → bool
Test an object and return true if it is an instance of Array

Examples

```
>>> array = Array()
>>> Array.is_list(array)
True
>>> Array.is_list({})
False
```

is_empty() → bool
Check if Array is empty

Examples

```
>>> Array().is_empty()
True
>>> Array(1, 2).is_empty()
False
```

static is_list(*arr: any*) → bool
Test an object and return true if it is an instance of list

Examples

```
>>> Array.is_list([])
True
>>> Array.is_list({})
False
```

map(*func*) → list
Function to create a new list based on callback function return

Parameters **func** (*function*) – A callback function that will be executed every iteration and should return something for reduce assemble new list.

Examples

```
>>> array = Array([1, 2, 3, 4])
>>> array.map(lambda item, index: item if item % 2 == 0 else None)
[2, 4]
```

pop(*pos: int = - 1*) → any
Removes the element at the specified position.

Parameters **pos** (*int, -1*) – Position of the element to be removed and returned.

Examples

```
>>> array = Array(1, 2, 3)
>>> array.pop()
3
>>> array.pop(0)
1
>>> array
<Array values=[2] >
```

reduce(*func*, *initial=[]*)
Function to create a new object based on callback function return

Parameters

- **func** (*function*) – A callback function that will be executed every iteration and should return something for reduce assemble new object
- **initial** (*any, []*) – Initial return value.

Raises **NotImplementedError** – func not defined or equal to None.

Examples

```
>>> array = Array([1, 2, 3, 4, 5, 6])
>>> def func(accumulator, item, index):
...     if item % 2 == 0:
...         return accumulator.append(item)
...     return
...
>>> array.reduce(func)
[2, 4, 6]
```

Note: if the callback function never returns anything, reduce will return the initial value itself

repetitions() → dict
Parses and returns all repetitions in the array.

Returns A dictionary of type item: int(repetitions)

Return type dict

Examples

```
>>> Array(*'Pyfunctools').repetitions()
{"P": 1, "y": 1, "f": 1, "u": 1, "n": 1, "c": 1, "t": 1, "o": 2, "l": 1, "s": 1}
>>> Array(*'Python').repetitions()
{"P": 1, "y": 1, "t": 1, "h": 1, "o": 1, "n": 1}
```

reverse()
Reverses the sorting order of the elements.

Examples

```
>>> array = Array(1, 2, 3)
>>> array.reverse()
>>> array
<Array values=[3, 2, 1] >
```

shift()
Removes the first element from an array and returns that removed element.

Examples

```
>>> array = Array(1, 2, 3)
>>> array.shift()
1
>>> array
<Array values=[2, 3] >
>>> array.shift()
2
>>> array
<Array values=[3] >
```

to_list() → list
Convert the array object to the builtin type list.

Note: If you prefer you can use list compression on the Array instance

unshift(**args*)
Adds one or more elements to the beginning of an array and returns a array modified.

pyfunctools.**get_version**(*release: bool = False*)
Get simple version or full version/release of pyfunc

Parameters **release** (*bool, False*) – if true, return full version of package

Examples

```
>>> get_version()
'0.1'
>>> get_version(True)
'0.1.0'
```

2.2 Submodules

2.2.1 pyfunctools.at module

pyfunctools.at.**at**(*obj: dict*, *path: str*) → any
Returns the value corresponding to path in obj

Parameters

- **obj** (*dict*) – The dictionary we want to get the value from
- **path** (*str*) – The path of the value that should be returned

Examples

```
>>> obj = { 'a': 1, 'b': { 'a': 1, 'b': [ 'a' ] } }
>>> at(obj, 'a')
1
>>> at(obj, 'b.a')
1
>>> at(obj, 'b.b')
[ 'a' ]
>>> at(obj, 'b.b.0')
'a'
>>> at(obj, 'b.b[0]')
'a'
```

2.2.2 pyfunctools.chunk module

pyfunctools.chunk.**chunk**(*arr: list*, *size: int = 1*) → list
This function takes a list and divides it into sublists of size equal to size.

Parameters

- **arr** (*list*) – list to split
- **size** (*int, optional*) – chunk size. Defaults to 1

Returns A new list containing the chunks of the original

Return type list

Examples

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

2.2.3 pyfunctools.compact module

pyfunctools.compact.**compact**(*arr: list*) → list

Create a new list with only the truthy values from the original.

Parameters **arr** (*list*) – original list

Returns a list with truthy values

Return type list

Examples

```
>>> compact([0, 1, 2, 3, '', None, False])
[1, 2, 3]
>>> compact([0, '', None, False])
[]
```

2.2.4 pyfunctools.filter module

pyfunctools.filter.**filter**(*arr: list*, *func*) → list

Filters items from a list based on callback function return

Parameters

- **arr** (*list*) – a list to iterate
- **func** (*function*) – a callback function

Examples

```
>>> array = Array(1, 2, 3, 4)
>>> array.filter(lambda item, index: item % 2 == 0)
[2, 4]
```

2.2.5 pyfunctools.flatten module

pyfunctools.flatten.**flatten**(*arr: list*, *level=1*) → list
Flat list.

Parameters

- **arr** (*list*) – original list
- **level** (*int | str*) – sublist level to planar

Note: Only accept whole levels or equal to 'all'

Raises **ValueError** – The level parameter entered is not integer or is different from 'all'

Examples

```
>>> flatten([1, [2, [3, [4, 5]]]])
[1, 2, [3, [4, 5]]]
>>> flatten([1, [2, [3, [4, 5]]]], 'all')
[1, 2, 3, 4, 5]
>>> flatten([1, [2, [3, [4, 5]]]], 0)
[1, [2, [3, [4, 5]]]]
```

2.2.6 pyfunctools.foreach module

pyfunctools.foreach.**forEach**(*arr: list*, *func*)
Iterates over a list and calls a function for each item, passing the item itself and the index

Parameters

- **arr** (*list*) – List to iterate
- **func** (*function*) – Callback function

Examples

```
>>> forEach([1, 2], lambda item, index: print(f'{item}, {index}'))
1, 0
2, 1
```

2.2.7 pyfunctools.map module

pyfunctools.map.**map**(*arr: list*, *func*) → list
Function to create a new list based on callback function return

Parameters

- **arr** (*list*) – a list to be iterated
- **func** (*function*) – a callback function that will be executed every iteration and should return something for reduce assemble new list.

Examples

```
>>> map([1, 2, 3, 4], lambda item, index: item if item % 2 == 0 else None)
[2, 4]
```

2.2.8 pyfunctools.memoize module

pyfunctools.memoize.**memoize**(*func*)

Creates a cache of the returns and arguments received by a function passed by parameter

Parameters **func** (*function*) – function to generate the cache

Raises **TypeError** – thrown when func is not a valid function

Examples

```
>>> fat = memoize(lambda n: 1 if n == 0 else n * fat(n-1))
>>> fat(4)
24
>>> @memoize
>>> def sums(*numbers):
        '''Receives a numeric sequence and calculates the sum of all numbers'''
        return Array(*numbers).reduce(lambda a, b, _: a+b, 0)
>>> sums(1, 2, 3, 4)
10
```

2.2.9 pyfunctools.pipeline module

pyfunctools.pipeline.**pipeline**(**funcs*)

Define a pipeline

Parameters ***funcs** (*list[callable]*) – a list of callables to be called later

Returns last pipe return

Examples

```
>>> pipes = pipeline(
...     lambda s: s.upper(),
...     lambda s: s + ' 95',
...     lambda s: s.replace(' ', '-')
... )
>>> pipes('functional python')
'FUNCTIONAL-PYTHON-95'
>>> pipes = pipeline(
...     lambda n: n + 1,
...     lambda x: x * 2
... )
>>> pipes(1)
4
```

2.2.10 pyfunctools.reduce module

pyfunctools.reduce.**reduce**(*arr: list*, *func*, *initial: any = []*)

Function to create a new object based on callback function return

Parameters

- **arr** (*list*) – A list to be iterated
- **func** (*function*) – A callback function that will be executed every iteration and should return something for reduce assemble new object
- **initial** (*any, []*) – Initial return value.

Raises **NotImplementedError** – Arr or func not defined or equal to None.

Examples

```
>>> arr = [1, 2, 3, 4, 5, 6]
>>> def func(accumulator, item, index):
...     if item % 2 == 0:
...         return accumulator.append(item)
...     return
...
>>> reduce(arr, func)
[2, 4, 6]
```

Note: if the callback function never returns anything, reduce will return the initial value itself

2.2.11 pyfunctools.utils module

pyfunctools.utils.**is_empty**(*value: any*) → bool

Checks if the value passed by parameter is empty.

Examples

```
>>> is_empty('')
True
>>> is_empty(None)
True
>>> is_empty([])
True
>>> is_empty(Array())
True
>>> is_empty({})
True
>>> is_empty(())
True
```

pyfunctools.utils.**is_equal**(*obj1*, *obj2*) → bool

Recursive function that checks if two parameters are equal

Examples

```
>>> is_equal(1, 1)
True
>>> is_equal('{}', '{}')
True
>>> is_equal({}, {})
True
>>> is_equal([], [])
True
>>> is_equal({'language': 'python'}, {'language': 'python'})
True
>>> is_equal({'language': 'python'}, {'language': 'js'})
False
>>> is_equal(Array(), Array())
True
```

pyfunctools.utils.**is_float**(*obj: any*) → bool

Tests an object and returns true if it is an int value.

Parameters **obj** (*any*) – Object to test

Examples

```
>>> is_float(1.0)
True
>>> is_float(10)
False
>>> is_float(lambda a: a)
False
>>> is_float('a')
False
```

pyfunctools.utils.**is_func**(*obj: any*) → bool

Tests an object and returns true if it is a function.

Parameters **obj** (*any*) – Object to test

Examples

```
>>> is_func(lambda a: a)
True
>>> is_func('a')
False
>>> is_func(10)
False
>>> def func():
...     pass
...
>>> is_func(func)
True
```

pyfunctools.utils.**is_int**(*obj: any*) → bool
Tests an object and returns true if it is an int value.

Parameters **obj** (*any*) – Object to test

Examples

```
>>> is_int(10)
True
>>> is_int(1.0)
False
>>> is_int(lambda a: a)
False
>>> is_int('a')
False
```

pyfunctools.utils.**is_negative**(*obj: Union[str, int, float]*) → bool
Check if number is negative

Examples

```
>>> is_negative('-2')
True
>>> is_negative('1')
False
>>> is_negative('1000')
False
>>> is_negative('-1000')
True
```

pyfunctools.utils.**is_num**(*obj: any*) → bool
Check if obj is number

Examples

```
>>> is_num(10)
True
>>> is_num(-10)
True
>>> is_num(+10)
True
>>> is_num(.10)
True
>>> is_num('.10')
True
>>> is_num('a')
False
```

pyfunctools.utils.**is_positive**(*obj: Union[str, int, float]*) → bool
Check if number is positive

Examples

```
>>> is_positive('1')
True
>>> is_positive('1000')
True
>>> is_positive('-2')
False
>>> is_positive('-1000')
False
```

pyfunctools.utils.**to_num**(*obj: any*) → Union[int, float]

Generic number converter

Parameters **obj** (*any*) – Will convert number notation to int or float

Raises **ValueError** – obj is not a number notation, it cannot be converted.

Examples

```
>>> to_num('10')
10
>>> to_num('1.0')
1.0
>>> to_num('.10')
0.1
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

R

S

T

U