pydash Documentation

Release 3.4.8

Derrick Gilland

January 06, 2017

Contents

The kitchen sink of Python utility libraries for doing "stuff" in a functional way. Based on the Lo-Dash Javascript library.

CHAPTER 1

Links

- Project: https://github.com/dgilland/pydash
- Documentation: http://pydash.readthedocs.org
- PyPi: https://pypi.python.org/pypi/pydash/
- TravisCI: https://travis-ci.org/dgilland/pydash

CHAPTER 2

Quickstart

The functions available from pydash can be used in two styles.

The first is by using the module directly or importing from it:

```
>>> import pydash
>>> from pydash import flatten

# Arrays
>>> flatten([1, 2, [3, [4, 5, [6, 7]]]])
[1, 2, 3, [4, 5, [6, 7]]]

>>> pydash.flatten_deep([1, 2, [3, [4, 5, [6, 7]]]])
[1, 2, 3, 4, 5, 6, 7]

# Collections
>>> pydash.pluck([{'name': 'moe', 'age': 40}, {'name': 'larry', 'age': 50}], 'name')
['moe', 'larry']

# Functions
>>> curried = pydash.curry(lambda a, b, c: a + b + c)
>>> curried(1, 2)(3)
6

# Objects
>>> pydash.omit({'name': 'moe', 'age': 40}, 'age')
{'name': 'moe'}

# Utilities
>>> pydash.times(lambda index: index, 3)
[0, 1, 2]

# Chaining
>>> pydash.chain([1, 2, 3, 4]).without(2, 3).reject(lambda x: x > 1).value()
[1]
```

The second style is to use the `py_` or `_` instances (they are the same object as two different aliases):

```
>>> from pydash import py_

# Method calling which is equivalent to pydash.flatten(...)
>>> py_.flatten([1, 2, [3, [4, 5, [6, 7]]]])
[1, 2, 3, [4, 5, [6, 7]]]

# Method chaining which is equivalent to pydash.chain(...)
```

```
>>> py_([1, 2, 3, 4]).without(2, 3).reject(lambda x: x > 1).value()
[1]

# Late method chaining
>>> py_().without(2, 3).reject(lambda x: x > 1)([1, 2, 3, 4])
[1]
```

See also:

For further details consult *API Reference*.

CHAPTER 3

Guide

3.1 Installation

pydash requires Python >= 2.6 or >= 3.3. It has no external dependencies.

To install from PyPi:

```
pip install pydash
```

3.2 Quickstart

The functions available from pydash can be used in two styles.

The first is by using the module directly or importing from it:

```
>>> import pydash
>>> from pydash import flatten

# Arrays
>>> flatten([1, 2, [3, [4, 5, [6, 7]]]])
[1, 2, 3, [4, 5, [6, 7]]]

>>> pydash.flatten_deep([1, 2, [3, [4, 5, [6, 7]]]])
[1, 2, 3, 4, 5, 6, 7]

# Collections
>>> pydash.pluck([{'name': 'moe', 'age': 40}, {'name': 'larry', 'age': 50}], 'name')
['moe', 'larry']

# Functions
>>> curried = pydash.curry(lambda a, b, c: a + b + c)
>>> curried(1, 2)(3)
6

# Objects
>>> pydash.omit({'name': 'moe', 'age': 40}, 'age')
{'name': 'moe'}

# Utilities
>>> pydash.times(lambda index: index, 3)
[0, 1, 2]
```

```
# Chaining
>>> pydash.chain([1, 2, 3, 4]).without(2, 3).reject(lambda x: x > 1).value()
[1]
```

The second style is to use the py_ or _ instances (they are the same object as two different aliases):

```
>>> from pydash import py_

# Method calling which is equivalent to pydash.flatten(...)
>>> py_.flatten([1, 2, [3, [4, 5, [6, 7]]]])
[1, 2, 3, [4, 5, [6, 7]]]

# Method chaining which is equivalent to pydash.chain(...)
>>> py_([1, 2, 3, 4]).without(2, 3).reject(lambda x: x > 1).value()
[1]

# Late method chaining
>>> py_().without(2, 3).reject(lambda x: x > 1)([1, 2, 3, 4])
[1]
```

See also:

For further details consult *API Reference*.

3.3 Lo-Dash Differences

3.3.1 Naming Conventions

pydash adheres to the following conventions:

- Function names use `snake_case` instead of `camelCase`.
- Any Lo-Dash function that shares its name with a reserved Python keyword will have an _ appended after it (e.g. `filter` in Lo-Dash would be `filter_` in pydash.
- Lo-Dash's `toArray()` is pydash's `to_list()`.
- Lo-Dash's `functions()` is pydash's `callables()`. This particular name difference was chosen in order to allow for the `functions.py` module file to exist at root of the project. Previously, `functions.py` existed in `pydash/api/` but in `v2.0.0`, it was decided to move everything in `api/` to `pydash/`. Therefore, In to avoid import ambiguities, the `functions()` function was renamed.

3.3.2 Callbacks

There are a few differences between extra callback style support:

- Lo-Dash's property style callback form uses shallow property access while the same form in pydash uses deep property access via the deep path string.
- Pydash has an explicit shallow property access of the form `['some_property']`.

3.3.3 Extra Functions

In addition to porting Lo-Dash, pydash contains functions found in lodashcontrib, lodashdeep, lodashmath, and underscorestring.

3.3.4 Function Behavior

Some of pydash's functions behave differently:

- *pydash.utilities.memoize()* uses all passed in arguments as the cache key by default instead of only using the first argument.

3.3.5 Templating

- pydash doesn't have `template()`. See *Templating* for more details.

3.4 Callbacks

For functions that support callbacks, there are several callback styles that can be used.

3.4.1 Callable Style

The most straight-forward callback is a regular callable object. For pydash functions that pass multiple arguments to their callback, the callable's argument signature does not need to support all arguments. Pydash's callback system will try to infer the number of supported arguments of the callable and only pass those arguments to the callback. However, there may be some edge cases where this will fail in which case one will need to wrap the callable in a `lambda` or `def ...` style function.

The arguments passed to most callbacks are:

```
callback(item, index, obj)
```

where `item` is an element of `obj`, `index` is the `dict` or `list` index, and `obj` is the original object being passed in. But not all callbacks support these arguments. Some functions support fewer callback arguments. See *API Reference* for more details.

```
>>> users = [
...     {'name': 'Michelangelo', 'active': False},
...     {'name': 'Donatello', 'active': False},
...     {'name': 'Leonardo', 'active': True}
... ]

# Single argument callback.
>>> callback = lambda item: item['name'] == 'Donatello'
>>> pydash.find_index(users, callback)
1

# Two argument callback.
>>> callback = lambda item, index: index == 3
>>> pydash.find_index(users, callback)
-1

# Three argument callback.
>>> callback = lambda item, index, obj: obj[index]['active']
>>> pydash.find_index(users, callback)
2
```

3.4.2 Shallow Property Style

The shallow property style callback is specified as a one item `list` containing the property value to return from an element. Internally, *pydash.utilities.prop()* is used to create the callback.

```
>>> users = [
...     {'name': 'Michelangelo', 'active': False},
...     {'name': 'Donatello', 'active': False},
...     {'name': 'Leonardo', 'active': True}
... ]
>>> pydash.find_index(users, ['active'])
2
```

3.4.3 Deep Property Style

The deep property style callback is specified as a deep property `string` of the nested object value to return from an element. Internally, *pydash.utilities.deep_prop()* is used to create the callback. See *Deep Path Strings* for more details.

```
>>> users = [
...     {'name': 'Michelangelo', 'location': {'city': 'Rome'}},
...     {'name': 'Donatello', 'location': {'city': 'Florence'}},
...     {'name': 'Leonardo', 'location': {'city': 'Amboise'}}
... ]
>>> pydash.collect(users, 'location.city')
['Rome', 'Florence', 'Amboise']
```

3.4.4 Matches Property Style

The matches property style callback is specified as a two item `list` containing a property key and value and returns `True` when an element's key is equal to value, else `False`. Internally, *pydash.utilities.matches_property()* is used to create the callback.

```
>>> users = [
...     {'name': 'Michelangelo', 'active': False},
...     {'name': 'Donatello', 'active': False},
...     {'name': 'Leonardo', 'active': True}
... ]
>>> pydash.find_index(users, ['active', False])
0
>>> pydash.find_last_index(users, ['active', False])
1
```

3.4.5 Matches Style

The matches style callback is specified as a `dict` object and returns `True` when an element matches the properties of the object, else `False`. Internally, *pydash.utilities.matches()* is used to create the callback.

```
>>> users = [
...     {'name': 'Michelangelo', 'location': {'city': 'Rome'}},
...     {'name': 'Donatello', 'location': {'city': 'Florence'}},
...     {'name': 'Leonardo', 'location': {'city': 'Amboise'}}
... ]
```

```
>>> pydash.collect(users, {'location': {'city': 'Florence'}})
[False, True, False]
```

3.5 Deep Path Strings

A deep path string is used to access a nested data structure of arbitrary length. Each level is separated by a `"."` and can be used on both dictionaries and lists. If a `"."` is contained in one of the dictionary keys, then it can be escaped using `"\"`. For accessing a dictionary key that is a number, it can be wrapped in brackets like `"[1]"`.

Examples:

```
>>> data = {'a': {'b': {'c': [0, 0, {'d': [0, {1: 2}]}]}}}
>>> pydash.deep_get(data, 'a.b.c.2.d.1.[1]')
2

>>> data = {'a': {'b.c.d': 2}}
>>> pydash.deep_get(data, r'a.b\.c\.d')
2
```

Functions that support deep path strings include:

- *pydash.collections.deep_pluck()*
- *pydash.objects.deep_get()*
- *pydash.objects.deep_has()*
- *pydash.objects.deep_set()*
- *pydash.utilities.deep_property()*/*pydash.utilities.deep_prop()*

Pydash's callback system also supports the deep property style callback using deep path strings.

3.6 Method Chaining

Method chaining in pydash is quite simple.

An initial value is provided:

```
from pydash import py_
py_([1, 2, 3, 4])

# Or through the chain() function
import pydash
pydash.chain([1, 2, 3, 4])
```

Methods are chained:

```
py_([1, 2, 3, 4]).without(2, 3).reject(lambda x: x > 1)
```

A final value is computed:

```
result = py_([1, 2, 3, 4]).without(2, 3).reject(lambda x: x > 1).value()
```

3.6.1 Lazy Evaluation

Method chaining is deferred (lazy) until `.value()` (or it's aliases `.value_of` or `.run()`) is called:

```
>>> from __future__ import print_function
>>> from pydash import py_

>>> def echo(value): print(value)

>>> lazy = py_([1, 2, 3, 4]).each(echo)

# None of the methods have been called yet.

>>> result = lazy.value()
1
2
3
4

# Each of the chained methods have now been called.

>>> assert result == [1, 2, 3, 4]

>>> result = lazy.run()
1
2
3
4
```

3.6.2 Committing a Chain

If one wishes to create a new chain object seeded with the computed value of another chain, then one can use the `commit` method:

```
>>> committed = lazy.commit()
1
2
3
4

>>> committed.value()
[1, 2, 3, 4]

>>> lazy.value()
1
2
3
4
[1, 2, 3, 4]
```

Committing is equivalent to:

```
committed = py_(lazy.value())
```

3.6.3 Late Value Passing

In *v3.0.0* the concept of late value passing was introduced to method chaining. This allows method chains to be re-used with different root values supplied. Essentially, ad-hoc functions can be created via the chaining syntax.

```
>>> square_sum = py_().power(2).sum()
>>> assert square_sum([1, 2, 3]) == 14
>>> assert square_sum([4, 5, 6]) == 77

>>> square_sum_square = square_sum.power(2)
>>> assert square_sum_square([1, 2, 3]) == 196
>>> assert square_sum_square([4, 5, 6]) == 5929
```

3.6.4 Planting a Value

To replace the initial value of a chain, use the `plant` method which will return a cloned chained using the new initial value:

```
>>> chained = py_([1, 2, 3, 4]).power(2).sum()
>>> chained.run()
30
>>> rechained = chained.plant([5, 6, 7, 8])
>>> rechained.run()
174
>>> chained.run()
30
```

3.6.5 Module Access

Another feature of the `py_` object, is that it provides module access to `pydash`:

```
>>> import pydash
>>> from pydash import py_

>>> assert py_.add is pydash.add
>>> py_.add([1, 2, 3]) == pydash.add([1, 2, 3])
True
```

Through `py_` any function that ends with `"_"` can be accessed without the trailing `"_"`:

```
>>> py_.filter([1, 2, 3], lambda x: x > 1) == pydash.filter_([1, 2, 3], lambda x: x > 1)
True
```

3.7 Templating

Templating has been purposely left out of pydash. Having a custom templating engine was never a goal of pydash even though Lo-Dash includes one. There already exist many mature and battle-tested templating engines like Jinja2 and Mako which are better suited to handling templating needs. However, if there was ever a strong request/justification for having templating in pydash (or a pull-request implementing it), then this decision could be re-evaluated.

3.8 Upgrading

3.8.1 From v2.x.x to v3.0.0

There were several breaking changes in `v3.0.0`:

- Make `to_string` convert `None` to empty string. (**breaking change**)
- Make the following functions work with empty strings and `None`: (**breaking change**)
  - `camel_case`
  - `capitalize`
  - `chars`
  - `chop`
  - `chop_right`
  - `class_case`
  - `clean`
  - `count_substr`
  - `decapitalize`
  - `ends_with`
  - `join`
  - `js_replace`
  - `kebab_case`
  - `lines`
  - `quote`
  - `re_replace`
  - `replace`
  - `series_phrase`
  - `series_phrase_serial`
  - `starts_with`
  - `surround`
- Reorder function arguments for `after` from `(n, func)` to `(func, n)`. (**breaking change**)
- Reorder function arguments for `before` from `(n, func)` to `(func, n)`. (**breaking change**)
- Reorder function arguments for `times` from `(n, callback)` to `(callback, n)`. (**breaking change**)
- Reorder function arguments for `js_match` from `(reg_exp, text)` to `(text, reg_exp)`. (**breaking change**)
- Reorder function arguments for `js_replace` from `(reg_exp, text, repl)` to `(text, reg_exp, repl)`. (**breaking change**)

And some potential breaking changes:

- Move `arrays.join` to `strings.join` (**possible breaking change**).

- Rename `join`/`implode`'s second parameter from `delimiter` to `separator`. (**possible breaking change**)
- Rename `split`/`explode`'s second parameter from `delimiter` to `separator`. (**possible breaking change**)

Some notable new features/functions:

- 31 new string methods
  - *pydash.strings.chars()*
  - *pydash.strings.chop()*
  - *pydash.strings.chop_right()*
  - pydash.strings.class_case()
  - *pydash.strings.clean()*
  - *pydash.strings.count_substr()*
  - *pydash.strings.decapitalize()*
  - *pydash.strings.has_substr()*
  - *pydash.strings.human_case()*
  - *pydash.strings.insert_substr()*
  - *pydash.strings.lines()*
  - *pydash.strings.number_format()*
  - *pydash.strings.pascal_case()*
  - *pydash.strings.predecessor()*
  - *pydash.strings.prune()*
  - *pydash.strings.re_replace()*
  - *pydash.strings.replace()*
  - *pydash.strings.separator_case()*
  - *pydash.strings.series_phrase()*
  - *pydash.strings.series_phrase_serial()*
  - *pydash.strings.slugify()*
  - *pydash.strings.split()*
  - *pydash.strings.strip_tags()*
  - *pydash.strings.substr_left()*
  - *pydash.strings.substr_left_end()*
  - *pydash.strings.substr_right()*
  - *pydash.strings.substr_right_end()*
  - *pydash.strings.successor()*
  - *pydash.strings.swap_case()*
  - *pydash.strings.title_case()*
  - *pydash.strings.unquote()*

- 1 new array method
  - *pydash.arrays.duplicates()*
- 2 new function methods
  - *pydash.functions.ary()*
  - *pydash.functions.rearg()*
- 1 new collection method:
  - *pydash.collections.sort_by_all()*
- 4 new object methods
  - *pydash.objects.to_boolean()*
  - *pydash.objects.to_dict()*
  - *pydash.objects.to_number()*
  - *pydash.objects.to_plain_object()*
- 4 new predicate methods
  - *pydash.predicates.is_blank()*
  - *pydash.predicates.is_builtin()* and alias *pydash.predicates.is_native()*
  - *pydash.predicates.is_match()*
  - *pydash.predicates.is_tuple()*
- 1 new utility method
  - *pydash.utilities.prop_of()* and alias *pydash.utilities.property_of()*
- 6 new aliases:
  - *pydash.predicates.is_bool()* for *pydash.predicates.is_boolean()*
  - *pydash.predicates.is_dict()* for *pydash.predicates.is_plain_object()*
  - *pydash.predicates.is_int()* for *pydash.predicates.is_integer()*
  - *pydash.predicates.is_num()* for *pydash.predicates.is_number()*
  - *pydash.strings.truncate()* for *pydash.strings.trunc()*
  - *pydash.strings.underscore_case()* for *pydash.strings.snake_case()*
- Chaining can now accept the root `value` argument late.
- Chains can be re-used with differnt initial values via `chain().plant`.
- New chains can be created using the chain's computed value as the new chain's initial value via `chain().commit`.
- Support iteration over class instance properties for non-list, non-dict, and non-iterable objects.

Late Value Chaining

The passing of the root `value` argument for chaining can now be done "late" meaning that you can build chains without providing a value at the beginning. This allows you to build a chain and re-use it with different root values:

```
>>> from pydash import py_

>>> square_sum = py_().power(2).sum()

>>> [square_sum([1, 2, 3]), square_sum([4, 5, 6]), square_sum([7, 8, 9])]
[14, 77, 194]
```

See also:

- For more details on method chaining, check out *Method Chaining*.
- For a full listing of changes in `v3.0.0`, check out the *Changelog*.

3.8.2 From v1.x.x to v2.0.0

There were several breaking and potentially breaking changes in `v2.0.0`:

- *`pydash.arrays.flatten()`* is now shallow by default. Previously, it was deep by default. For deep flattening, use either `flatten(..., is_deep=True)` or `flatten_deep(...)`.
- *`pydash.predicates.is_number()`* now returns `False` for boolean `True` and `False`. Previously, it returned `True`.
- Internally, the files located in `pydash.api` were moved to `pydash`. If you imported from `pydash.api.<module>`, then it's recommended to change your imports to pull from `pydash`.
- The function `functions()` was renamed to `callables()` to avoid ambiguities with the module `functions.py`.

Some notable new features:

- Callback functions no longer require the full call signature definition. See *Callbacks* for more details.
- A new "_" instance was added which supports both method chaining and module method calling. See *py_ Instance* for more details.

See also:

For a full listing of changes in `v2.0.0`, check out the *Changelog*.

CHAPTER 4

API Reference

Includes links to source code.

4.1 API Reference

All public functions are available from the main module.

```
import pydash

pydash.<function>
```

This is the recommended way to use pydash.

```
# OK (importing main module)
import pydash
pydash.where({})

# OK (import from main module)
from pydash import where
where({})

# NOT RECOMMENDED (importing from submodule)
from pydash.collections import where
```

Only the main pydash module API is guaranteed to adhere to semver. It's possible that backwards incompatibility outside the main module API could be broken between minor releases.

4.1.1 py_ Instance

There is a special `py_` instance available from `pydash` that supports method calling and method chaining from a single object:

```
from pydash import py_

# Method calling
py_.initial([1, 2, 3, 4, 5]) == [1, 2, 3, 4]

# Method chaining
py_([1, 2, 3, 4, 5]).initial().value() == [1, 2, 3, 4]

# Method aliasing to underscore suffixed methods that shadow builtin names
```

```
py_.map is py_.map_
py_([1, 2, 3]).map(_.to_string).value() == py_([1, 2, 3]).map_(_.to_string).value()
```

The py_ instance is basically a combination of using pydash.<function> and pydash.chain.

A full listing of aliased py_ methods:

- _.object is *pydash.arrays.object_()*
- _.slice is *pydash.arrays.slice_()*
- _.zip is *pydash.arrays.zip_()*
- _.all is *pydash.collections.all_()*
- _.any is *pydash.collections.any_()*
- _.filter is *pydash.collections.filter_()*
- _.map is *pydash.collections.map_()*
- _.max is *pydash.collections.max_()*
- _.min is *pydash.collections.min_()*
- _.reduce is *pydash.collections.reduce_()*
- _.pow is *pydash.numerical.pow_()*
- _.round is *pydash.numerical.round_()*
- _.sum is *pydash.numerical.sum_()*
- _.property is *pydash.utilities.property_()*
- _.range is *pydash.utilities.range_()*

4.1.2 Arrays

Functions that operate on lists.

New in version 1.0.0.

pydash.arrays.**append**(*array*, **items*)

Push items onto the end of *array* and return modified *array*.

Parameters

- **array** (*list*) – List to push to.
- **items** (*mixed*) – Items to append.

Returns Modified *array*.

Return type list

Warning: *array* is modified in place.

Example

```
>>> array = [1, 2, 3]
>>> push(array, 4, 5, [6])
[1, 2, 3, 4, 5, [6]]
```

See also:

- *push()* (main definition)
- *append()* (alias)

New in version 2.2.0.

pydash.arrays.**cat**(**arrays*)
Concatenates zero or more lists into one.

Parameters **arrays** (*list*) – Lists to concatenate.

Returns Concatenated list.

Return type list

Example

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

New in version 2.0.0.

pydash.arrays.**chunk**(*array*, *size=1*)
Creates a list of elements split into groups the length of *size*. If *array* can't be split evenly, the final chunk will be the remaining elements.

Parameters

- **array** (*list*) – List to chunk.
- **size** (*int, optional*) – Chunk size. Defaults to 1.

Returns New list containing chunks of *array*.

Return type list

Example

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

New in version 1.1.0.

pydash.arrays.**compact**(*array*)
Creates a list with all falsey values of array removed.

Parameters **array** (*list*) – List to compact.

Returns Compacted list.

Return type list

Example

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

New in version 1.0.0.

pydash.arrays.**concat**(**arrays*)

Concatenates zero or more lists into one.

Parameters **arrays** (*list*) – Lists to concatenate.

Returns Concatenated list.

Return type list

Example

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

New in version 2.0.0.

pydash.arrays.**difference**(*array*, **lists*)

Creates a list of list elements not present in the other lists.

Parameters

- **array** (*list*) – List to process.
- **lists** (*list*) – Lists to check.

Returns Difference of the lists.

Return type list

Example

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

New in version 1.0.0.

pydash.arrays.**drop**(*array*, *n=1*)

Creates a slice of *array* with *n* elements dropped from the beginning.

Parameters

- **array** (*list*) – List to process.
- **n** (*int, optional*) – Number of elements to drop. Defaults to `1`.

Returns Dropped list.

Return type list

Example

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

New in version 1.0.0.

Changed in version 1.1.0: Added `n` argument and removed as alias of *rest()*.

Changed in version 3.0.0: Made `n` default to `1`.

pydash.arrays.**drop_right**(*array*, *n=1*)

Creates a slice of *array* with *n* elements dropped from the end.

Parameters

- **array** (*list*) – List to process.
- **n** (*int, optional*) – Number of elements to drop. Defaults to `1`.

Returns Dropped list.

Return type list

Example

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

New in version 1.1.0.

Changed in version 3.0.0: Made `n` default to `1`.

pydash.arrays.**drop_right_while**(*array*, *callback=None*)

Creates a slice of *array* excluding elements dropped from the end. Elements are dropped until the *callback* returns falsey. The *callback* is invoked with three arguments: `(value, index, array)`.

Parameters

- **array** (*list*) – List to process.
- **callback** (*mixed*) – Callback called per iteration

Returns Dropped list.

Return type list

Example

```
>>> drop_right_while([1, 2, 3, 4], lambda x: x >= 3)
[1, 2]
```

New in version 1.1.0.

pydash.arrays.**drop_while**(*array*, *callback=None*)

Creates a slice of *array* excluding elements dropped from the beginning. Elements are dropped until the *callback* returns falsey. The *callback* is invoked with three arguments: `(value, index, array)`.

Parameters

- **array** (*list*) – List to process.
- **callback** (*mixed*) – Callback called per iteration

Returns Dropped list.

Return type list

Example

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

New in version 1.1.0.

pydash.arrays.**duplicates**(*array*, *callback=None*)

Creates a unique list of duplicate values from *array*. If callback is passed, each element of array is passed through a callback before duplicates are computed. The callback is invoked with three arguments: `(value, index, array)`. If a property name is passed for callback, the created *pydash.collections.pluck()* style callback will return the property value of the given element. If an object is passed for callback, the created *pydash.collections.where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **array** (*list*) – List to process.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns List of duplicates.

Return type list

Example

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

New in version 3.0.0.

pydash.arrays.**fill**(*array*, *value*, *start=0*, *end=None*)

Fills elements of array with value from start up to, but not including, end.

Parameters

- **array** (*list*) – List to fill.
- **value** (*mixed*) – Value to fill with.
- **start** (*int, optional*) – Index to start filling. Defaults to `0`.
- **end** (*int, optional*) – Index to end filling. Defaults to `len(array)`.

Returns Filled *array*.

Return type list

Example

```
>>> fill([1, 2, 3, 4, 5], 0)
[0, 0, 0, 0, 0]
>>> fill([1, 2, 3, 4, 5], 0, 1, 3)
[1, 0, 0, 4, 5]
>>> fill([1, 2, 3, 4, 5], 0, 0, 100)
[0, 0, 0, 0, 0]
```

Warning: *array* is modified in place.

New in version 3.1.0.

pydash.arrays.**find_index**(*array*, *callback=None*)

This method is similar to *pydash.collections.find()*, except that it returns the index of the element that passes the callback check, instead of the element itself.

Parameters

- **array** (*list*) – List to process.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Index of found item or `-1` if not found.

Return type int

Example

```
>>> find_index([1, 2, 3, 4], lambda x: x >= 3)
2
>>> find_index([1, 2, 3, 4], lambda x: x > 4)
-1
```

New in version 1.0.0.

pydash.arrays.**find_last_index**(*array*, *callback=None*)

This method is similar to *find_index()*, except that it iterates over elements from right to left.

Parameters

- **array** (*list*) – List to process.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Index of found item or `-1` if not found.

Return type int

Example

```
>>> find_last_index([1, 2, 3, 4], lambda x: x >= 3)
3
>>> find_index([1, 2, 3, 4], lambda x: x > 4)
-1
```

New in version 1.0.0.

pydash.arrays.**first**(*array*)

Return the first element of *array*.

Parameters **array** (*list*) – List to process.

Returns First element of list.

Return type mixed

Example

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

See also:

- *first()* (main definition)
- *head()* (alias)
- *take()* (alias)

New in version 1.0.0.

pydash.arrays.**flatten**(*array*, *is_deep=False*)

Flattens a nested array. If *is_deep* is `True` the array is recursively flattened, otherwise it is only flattened a single level.

Parameters

- **array** (*list*) – List to process.
- **is_deep** (*bool, optional*) – Whether to recursively flatten *array*.

Returns Flattened list.

Return type list

Example

```
>>> flatten([[1], [2, [3]], [[4]]])
[1, 2, [3], [4]]
>>> flatten([[1], [2, [3]], [[4]]], True)
[1, 2, 3, 4]
```

New in version 1.0.0.

Changed in version 2.0.0: Removed `callback` option. Added `is_deep` option. Made it shallow by default.

pydash.arrays.**flatten_deep**(*array*)

Flattens a nested array recursively. This is the same as calling `flatten(array, is_deep=True)`.

Parameters **array** (*list*) – List to process.

Returns Flattened list.

Return type list

Example

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

New in version 2.0.0.

pydash.arrays.**head**(*array*)

Return the first element of *array*.

Parameters **array** (*list*) – List to process.

Returns First element of list.

Return type mixed

Example

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

See also:

- *first()* (main definition)
- *head()* (alias)
- *take()* (alias)

New in version 1.0.0.

pydash.arrays.**index_of**(*array*, *value*, *from_index=0*)

Gets the index at which the first occurrence of value is found.

Parameters

- **array** (*list*) – List to search.
- **value** (*mixed*) – Value to search for.
- **from_index** (*int, optional*) – Index to search from.

Returns Index of found item or `-1` if not found.

Return type int

Example

```
>>> index_of([1, 2, 3, 4], 2)
1
>>> index_of([2, 1, 2, 3], 2, from_index=1)
2
```

New in version 1.0.0.

pydash.arrays.**initial**(*array*)

Return all but the last element of *array*.

Parameters **array** (*list*) – List to process.

Returns Initial part of *array*.

Return type list

Example

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

New in version 1.0.0.

pydash.arrays.**intercalate**(*array*, *separator*)
Like *intersperse()* for lists of lists but shallowly flattening the result.

Parameters

- **array** (*list*) – List to intercalate.
- **separator** (*mixed*) – Element to insert.

Returns Intercalated list.

Return type list

Example

```
>>> intercalate([1, [2], [3], 4], 'x')
[1, 'x', 2, 'x', 3, 'x', 4]
```

New in version 2.0.0.

pydash.arrays.**interleave**(**arrays*)
Merge multiple lists into a single list by inserting the next element of each list by sequential round-robin into the new list.

Parameters **arrays** (*list*) – Lists to interleave.

Retruns: list: Interleaved list.

Example

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

New in version 2.0.0.

pydash.arrays.**intersection**(**arrays*)
Computes the intersection of all the passed-in arrays.

Parameters **arrays** (*list*) – Lists to process.

Returns Intersection of provided lists.

Return type list

Example

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

New in version 1.0.0.

pydash.arrays.**intersperse**(*array*, *separator*)
Insert a separating element between the elements of *array*.

Parameters

- **array** (*list*) – List to intersperse.
- **separator** (*mixed*) – Element to insert.

Returns Interspersed list.

Return type list

Example

```
>>> intersperse([1, [2], [3], 4], 'x')
[1, 'x', [2], 'x', [3], 'x', 4]
```

New in version 2.0.0.

pydash.arrays.**last**(*array*)

Return the last element of *array*.

Parameters **array** (*list*) – List to process.

Returns Last part of *array*.

Return type mixed

Example

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

New in version 1.0.0.

pydash.arrays.**last_index_of**(*array*, *value*, *from_index=None*)

Gets the index at which the last occurrence of value is found.

Parameters

- **array** (*list*) – List to search.
- **value** (*mixed*) – Value to search for.
- **from_index** (*int, optional*) – Index to search from.

Returns Index of found item or `False` if not found.

Return type int

Example

```
>>> last_index_of([1, 2, 2, 4], 2)
2
>>> last_index_of([1, 2, 2, 4], 2, from_index=1)
1
```

New in version 1.0.0.

pydash.arrays.**mapcat**(*array*, *callback=None*)

Map a callback to each element of a list and concatenate the results into a single list using *cat()*.

Parameters

- **array** (*list*) – List to map and concatenate.
- **callback** (*mixed*) – Callback to apply to each element.

Returns Mapped and concatenated list.

Return type list

Example

```
>>> mapcat(range(4), lambda x: list(range(x)))
[0, 0, 1, 0, 1, 2]
```

New in version 2.0.0.

pydash.arrays.**object_**(*keys*, *values=None*)

Creates a dict composed from lists of keys and values. Pass either a single two dimensional list, i.e. `[[key1, value1], [key2, value2]]`, or two lists, one of keys and one of corresponding values.

Parameters

- **keys** (*list*) – Either a list of keys or a list of `[key, value]` pairs
- **values** (*list, optional*) – List of values to zip

Returns Zipped dict.

Return type dict

Example

```
>>> zip_object([1, 2, 3], [4, 5, 6])
{1: 4, 2: 5, 3: 6}
```

See also:

- *zip_object()* (main definition)
- *object_()* (alias)

New in version 1.0.0.

pydash.arrays.**pull**(*array*, **values*)

Removes all provided values from the given array.

Parameters

- **array** (*list*) – List to pull from.
- **values** (*mixed*) – Values to remove.

Returns Modified *array*.

Return type list

Warning: *array* is modified in place.

Example

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

New in version 1.0.0.

pydash.arrays.**pull_at**(*array*, **indexes*)

Removes elements from *array* corresponding to the specified indexes and returns a list of the removed elements. Indexes may be specified as a list of indexes or as individual arguments.

Parameters

- **array** (*list*) – List to pull from.
- **indexes** (*int*) – Indexes to pull.

Returns Modified *array*.

Return type list

Warning: *array* is modified in place.

Example

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

New in version 1.1.0.

pydash.arrays.**push**(*array*, **items*)

Push items onto the end of *array* and return modified *array*.

Parameters

- **array** (*list*) – List to push to.
- **items** (*mixed*) – Items to append.

Returns Modified *array*.

Return type list

Warning: *array* is modified in place.

Example

```
>>> array = [1, 2, 3]
>>> push(array, 4, 5, [6])
[1, 2, 3, 4, 5, [6]]
```

See also:

- *push()* (main definition)
- *append()* (alias)

New in version 2.2.0.

pydash.arrays.**remove**(*array*, *callback=None*)

Removes all elements from a list that the callback returns truthy for and returns an array of removed elements.

Parameters

- **array** (*list*) – List to remove elements from.

- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Removed elements of *array*.

Return type list

Warning: *array* is modified in place.

Example

```
>>> array = [1, 2, 3, 4]
>>> items = remove(array, lambda x: x >= 3)
>>> items
[3, 4]
>>> array
[1, 2]
```

New in version 1.0.0.

pydash.arrays.**rest**(*array*)

Return all but the first element of *array*.

Parameters **array** (*list*) – List to process.

Returns Rest of the list.

Return type list

Example

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

See also:

- *rest()* (main definition)
- *tail()* (alias)

New in version 1.0.0.

pydash.arrays.**reverse**(*array*)

Return *array* in reverse order.

Parameters **array** (*list|string*) – Object to process.

Returns Reverse of object.

Return type list|string

Example

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

New in version 2.2.0.

pydash.arrays.**shift**(*array*)

Remove the first element of *array* and return it.

Parameters **array** (*list*) – List to shift.

Returns First element of *array*.

Return type mixed

Warning: *array* is modified in place.

Example

```
>>> array = [1, 2, 3, 4]
>>> item = shift(array)
>>> item
1
>>> array
[2, 3, 4]
```

New in version 2.2.0.

pydash.arrays.**slice_**(*array*, *start=0*, *end=None*)

Slices *array* from the *start* index up to, but not including, the *end* index.

Parameters

- **array** (*list*) – Array to slice.
- **start** (*int, optional*) – Start index. Defaults to `0`.
- **end** (*int, optional*) – End index. Defaults to selecting the value at `start` index.

Returns Sliced list.

Return type list

Example

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

New in version 1.1.0.

pydash.arrays.**sort**(*array*, *comparison=None*, *key=None*, *reverse=False*)

Sort *array* using optional *comparison*, *key*, and *reverse* options and return sorted *array*.

Note: Python 3 removed the option to pass a custom comparison function and instead only allows a key function. Therefore, if a comparison function is passed in, it will be converted to a key function automatically using `functools.cmp_to_key`.

Parameters

- **array** (*list*) – List to sort.
- **comparison** (*callable, optional*) – A custom comparison function used to sort the list. Function should accept two arguments and return a negative, zero, or position number depending on whether the first argument is considered smaller than, equal to, or larger than the second argument. Defaults to `None`. This argument is mutually exclusive with *key*.
- **key** (*callback, optional*) – A function of one argument used to extract a a comparison key from each list element. Defaults to `None`. This argument is mutually exclusive with *comparison*.
- **reverse** (*bool, optional*) – Whether to reverse the sort. Defaults to `False`.

Returns Sorted list.

Return type list

Warning: *array* is modified in place.

Example

```
>>> sort([2, 1, 4, 3])
[1, 2, 3, 4]
>>> sort([2, 1, 4, 3], reverse=True)
[4, 3, 2, 1]
>>> results = sort([{'a': 2, 'b': 1},                                {'a': 3, 'b': 2},
>>> assert results == [{'a': 0, 'b': 3},                                {'a': 2, 'b': 1},
```

New in version 2.2.0.

pydash.arrays.**sorted_index**(*array*, *value*, *callback=None*)

Determine the smallest index at which *value* should be inserted into array in order to maintain the sort order of the sorted array. If callback is passed, it will be executed for value and each element in array to compute their sort ranking. The callback is invoked with one argument: `(value)`. If a property name is passed for callback, the created *pydash.collections.pluck()* style callback will return the property value of the given element. If an object is passed for callback, the created *pydash.collections.where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **array** (*list*) – List to inspect.
- **value** (*mixed*) – Value to evaluate.
- **callback** (*mixed, optional*) – Callback to determine sort key.

Returns Smallest index.

Return type int

Example

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

New in version 1.0.0.

pydash.arrays.**sorted_last_index**(*array*, *value*, *callback=None*)

This method is like *sorted_index()* except that it returns the highest index at which a value should be inserted into a given sorted array in order to maintain the sort order of the array.

Parameters

- **array** (*list*) – List to inspect.
- **value** (*mixed*) – Value to evaluate.
- **callback** (*mixed, optional*) – Callback to determine sort key.

Returns Highest index.

Return type int

Example

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

New in version 1.1.0.

pydash.arrays.**splice**(*array*, *index*, *how_many=None*, **items*)

Modify the contents of *array* by inserting elements starting at *index* and removing *how_many* number of elements after *index*.

Parameters

- **array** (*list|str*) – List to splice.
- **index** (*int*) – Index to splice at.
- **how_many** (*int, optional*) – Number of items to remove starting at *index*. If `None` then all items after *index* are removed. Defaults to `None`.
- **items** (*mixed*) – Elements to insert starting at *index*. Each item is inserted in the order given.

Returns The removed elements of *array* or the spliced string.

Return type list|str

Warning: *array* is modified in place if `list`.

Example

```
>>> array = [1, 2, 3, 4]
>>> splice(array, 1)
[2, 3, 4]
>>> array
[1]
>>> array = [1, 2, 3, 4]
>>> splice(array, 1, 2)
[2, 3]
>>> array
[1, 4]
>>> array = [1, 2, 3, 4]
>>> splice(array, 1, 2, 0, 0)
```

```
    [2, 3]
    >>> array
    [1, 0, 0, 4]
```

New in version 2.2.0.

Changed in version 3.0.0: Support string splicing.

pydash.arrays.**split_at**(*array*, *index*)

Returns a list of two lists composed of the split of *array* at *index*.

Parameters

- **array** (*list*) – List to split.
- **index** (*int*) – Index to split at.

Returns Split list.

Return type list

Example

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

New in version 2.0.0.

pydash.arrays.**tail**(*array*)

Return all but the first element of *array*.

Parameters **array** (*list*) – List to process.

Returns Rest of the list.

Return type list

Example

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

See also:

- *rest()* (main definition)
- *tail()* (alias)

New in version 1.0.0.

pydash.arrays.**take**(*array*, *n=1*)

Creates a slice of *array* with *n* elements taken from the beginning.

Parameters

- **array** (*list*) – List to process.
- **n** (*int, optional*) – Number of elements to take. Defaults to `1`.

Returns Taken list.

Return type list

Example

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

New in version 1.0.0.

Changed in version 1.1.0: Added `n` argument and removed as alias of *`first()`*.

Changed in version 3.0.0: Made `n` default to `1`.

`pydash.arrays.`**`take_right`**`(`*array*, *n=1*`)`

Creates a slice of *array* with *n* elements taken from the end.

Parameters

- **`array`** (*`list`*) – List to process.
- **`n`** (*`int, optional`*) – Number of elements to take. Defaults to `1`.

Returns Taken list.

Return type list

Example

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

New in version 1.1.0.

Changed in version 3.0.0: Made `n` default to `1`.

`pydash.arrays.`**`take_right_while`**`(`*array*, *callback=None*`)`

Creates a slice of *array* with elements taken from the end. Elements are taken until the *callback* returns falsey. The *callback* is invoked with three arguments: `(value, index, array)`.

Parameters

- **`array`** (*`list`*) – List to process.
- **`callback`** (*`mixed`*) – Callback called per iteration

Returns Dropped list.

Return type list

Example

```
>>> take_right_while([1, 2, 3, 4], lambda x: x >= 3)
[3, 4]
```

New in version 1.1.0.

`pydash.arrays.`**`take_while`**`(`*array*, *callback=None*`)`

Creates a slice of *array* with elements taken from the beginning. Elements are taken until the *callback* returns falsey. The *callback* is invoked with three arguments: `(value, index, array)`.

Parameters

- **`array`** (*`list`*) – List to process.

- **callback** (*mixed*) – Callback called per iteration

Returns Taken list.

Return type list

Example

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

New in version 1.1.0.

pydash.arrays.**union**(**arrays*)

Computes the union of the passed-in arrays.

Parameters **arrays** (*list*) – Lists to unionize.

Returns Unionized list.

Return type list

Example

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

New in version 1.0.0.

pydash.arrays.**uniq**(*array*, *callback=None*)

Creates a duplicate-value-free version of the array. If callback is passed, each element of array is passed through a callback before uniqueness is computed. The callback is invoked with three arguments: `(value, index, array)`. If a property name is passed for callback, the created *pydash.collections.pluck()* style callback will return the property value of the given element. If an object is passed for callback, the created *pydash.collections.where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **array** (*list*) – List to process.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Unique list.

Return type list

Example

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

See also:

- *uniq()* (main definition)
- *unique()* (alias)

New in version 1.0.0.

pydash.arrays.**unique**(*array*, *callback=None*)

Creates a duplicate-value-free version of the array. If callback is passed, each element of array is passed through a callback before uniqueness is computed. The callback is invoked with three arguments: `(value, index, array)`. If a property name is passed for callback, the created *pydash.collections.pluck()* style callback will return the property value of the given element. If an object is passed for callback, the created *pydash.collections.where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **array** (*list*) – List to process.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Unique list.

Return type list

Example

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

See also:

- *uniq()* (main definition)
- *unique()* (alias)

New in version 1.0.0.

pydash.arrays.**unshift**(*array*, **items*)

Insert the given elements at the beginning of *array* and return the modified list.

Parameters

- **array** (*list*) – List to modify.
- **items** (*mixed*) – Items to insert.

Returns Modified list.

Return type list

Warning: *array* is modified in place.

Example

```
>>> array = [1, 2, 3, 4]
>>> unshift(array, -1, -2)
[-1, -2, 1, 2, 3, 4]
>>> array
[-1, -2, 1, 2, 3, 4]
```

New in version 2.2.0.

pydash.arrays.**unzip**(*array*)

The inverse of *zip_()*, this method splits groups of elements into lists composed of elements from each group at their corresponding indexes.

Parameters **array** (*list*) – List to process.

Returns Unzipped list.

Return type list

Example

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

New in version 1.0.0.

pydash.arrays.**unzip_with**(*array*, *callback=None*)

This method is like *unzip()* except that it accepts a callback to specify how regrouped values should be combined. The callback is invoked with four arguments: `(accumulator, value, index, group)`.

Parameters

- **array** (*list*) – List to process.
- **callback** (*callable, optional*) – Function to combine regrouped values.

Returns Unzipped list.

Return type list

Example

```
>>> from pydash import add
>>> unzip_with([[1, 10, 100], [2, 20, 200]], add)
[3, 30, 300]
```

New in version 3.3.0.

pydash.arrays.**without**(*array*, **values*)

Creates an array with all occurrences of the passed values removed.

Parameters

- **array** (*list*) – List to filter.
- **values** (*mixed*) – Values to remove.

Returns Filtered list.

Return type list

Example

```
>>> without([1, 2, 3, 2, 4, 4], 2, 4)
[1, 3]
```

New in version 1.0.0.

pydash.arrays.**xor**(*array*, **lists*)

Creates a list that is the symmetric difference of the provided lists.

Parameters

- **array** (*list*) – List to process.
- ***lists** (*list*) – Lists to xor with.

Returns XOR'd list.

Return type list

Example

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

New in version 1.0.0.

pydash.arrays.**zip_**(**arrays*)

Groups the elements of each array at their corresponding indexes. Useful for separate data sources that are coordinated through matching array indexes.

Parameters **arrays** (*list*) – Lists to process.

Returns Zipped list.

Return type list

Example

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

New in version 1.0.0.

pydash.arrays.**zip_object**(*keys*, *values=None*)

Creates a dict composed from lists of keys and values. Pass either a single two dimensional list, i.e. `[[key1, value1], [key2, value2]]`, or two lists, one of keys and one of corresponding values.

Parameters

- **keys** (*list*) – Either a list of keys or a list of `[key, value]` pairs
- **values** (*list, optional*) – List of values to zip

Returns Zipped dict.

Return type dict

Example

```
>>> zip_object([1, 2, 3], [4, 5, 6])
{1: 4, 2: 5, 3: 6}
```

See also:

- *zip_object()* (main definition)

- *object_()* (alias)

New in version 1.0.0.

pydash.arrays.**zip_with**(**arrays*, ***kargs*)

This method is like `zip()` except that it accepts a callback to specify how grouped values should be combined. The callback is invoked with four arguments: `(accumulator, value, index, group)`.

Parameters

- ***arrays** (*list*) – Lists to process.
- **callback** (*function*) – Function to combine grouped values.

Returns Zipped list of grouped elements.

Return type list

Example

```
>>> from pydash import add
>>> zip_with([1, 2], [10, 20], [100, 200], add)
[111, 222]
>>> zip_with([1, 2], [10, 20], [100, 200], callback=add)
[111, 222]
```

New in version 3.3.0.

4.1.3 Chaining

Method chaining interface.

New in version 1.0.0.

pydash.chaining.**chain**(*value=<pydash.helpers._NoValue object>*)

Creates a `Chain` object which wraps the given value to enable intuitive method chaining. Chaining is lazy and won't compute a final value until `Chain.value()` is called.

Parameters **value** (*mixed*) – Value to initialize chain operations with.

Returns Instance of `Chain` initialized with *value*.

Return type `Chain`

Example

```
>>> chain([1, 2, 3, 4]).map(lambda x: x * 2).sum().value()
20
>>> chain().map(lambda x: x * 2).sum()([1, 2, 3, 4])
20
```

```
>>> summer = chain([1, 2, 3, 4]).sum()
>>> new_summer = summer.plant([1, 2])
>>> new_summer.value()
3
>>> summer.value()
10
```

```
>>> def echo(item): print(item)
>>> summer = chain([1, 2, 3, 4]).each(echo).sum()
>>> committed = summer.commit()
1
2
3
4
>>> committed.value()
10
>>> summer.value()
1
2
3
4
10
```

New in version 1.0.0.

Changed in version 2.0.0: Made chaining lazy.

Changed in version 3.0.0: - Added support for late passing of *value*. - Added `Chain.plant()` for replacing initial chain value. - Added `Chain.commit()` for returning a new `Chain` instance initialized with the results from calling `Chain.value()`.

`pydash.chaining.`**`tap`**(*value*, *interceptor*)

Invokes *interceptor* with the *value* as the first argument and then returns *value*. The purpose of this method is to "tap into" a method chain in order to perform operations on intermediate results within the chain.

Parameters

- **`value`** (*`mixed`*) – Current value of chain operation.
- **`interceptor`** (*`function`*) – Function called on *value*.

Returns *value* after *interceptor* call.

Return type mixed

Example

```
>>> data = []
>>> def log(value): data.append(value)
>>> chain([1, 2, 3, 4]).map(lambda x: x * 2).tap(log).value()
[2, 4, 6, 8]
>>> data
[[2, 4, 6, 8]]
```

New in version 1.0.0.

`pydash.chaining.`**`thru`**(*value*, *interceptor*)

Returns the result of calling *interceptor* on *value*. The purpose of this method is to pass *value* through a function during a method chain.

Parameters

- **`value`** (*`mixed`*) – Current value of chain operation.
- **`interceptor`** (*`function`*) – Function called with *value*.

Returns Results of `interceptor(value)`.

Return type mixed

Example

```
>>> chain([1, 2, 3, 4]).thru(lambda x: x * 2).value()
[1, 2, 3, 4, 1, 2, 3, 4]
```

New in version 2.0.0.

4.1.4 Collections

Functions that operate on lists and dicts.

New in version 1.0.0.

pydash.collections.**all_**(*collection*, *callback=None*)

Checks if the callback returns a truthy value for all elements of a collection. The callback is invoked with three arguments: `(value, index|key, collection)`. If a property name is passed for callback, the created *pluck()* style callback will return the property value of the given element. If an object is passed for callback, the created *where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Whether all elements are truthy.

Return type bool

Example

```
>>> every([1, True, 'hello'])
True
>>> every([1, False, 'hello'])
False
>>> every([{'a': 1}, {'a': True}, {'a': 'hello'}], 'a')
True
>>> every([{'a': 1}, {'a': False}, {'a': 'hello'}], 'a')
False
>>> every([{'a': 1}, {'a': 1}], {'a': 1})
True
>>> every([{'a': 1}, {'a': 2}], {'a': 1})
False
```

See also:

- *every()* (main definition)
- *all_()* (alias)

New in version 1.0.0.

pydash.collections.**any_**(*collection*, *callback=None*)

Checks if the callback returns a truthy value for any element of a collection. The callback is invoked with three arguments: `(value, index|key, collection)`. If a property name is passed for callback, the created *pluck()* style callback will return the property value of the given element. If an object is passed for callback,

the created *where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callbacked** (*mixed, optional*) – Callback applied per iteration.

Returns Whether any of the elements are truthy.

Return type bool

Example

```
>>> some([False, True, 0])
True
>>> some([False, 0, None])
False
>>> some([1, 2, 3, 4], lambda x: x >= 3)
True
>>> some([1, 2, 3, 4], lambda x: x == 0)
False
```

See also:

- *some()* (main definition)
- *any_()* (alias)

New in version 1.0.0.

pydash.collections.**at**(*collection*, **indexes*)

Creates a list of elements from the specified indexes, or keys, of the collection. Indexes may be specified as individual arguments or as arrays of indexes.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **indexes** (*mixed*) – The indexes of *collection* to retrieve, specified as individual indexes or arrays of indexes.

Returns filtered list

Return type list

Example

```
>>> at([1, 2, 3, 4], 0, 2)
[1, 3]
>>> at({'a': 1, 'b': 2, 'c': 3, 'd': 4}, 'a', 'c')
[1, 3]
```

New in version 1.0.0.

pydash.collections.**collect**(*collection*, *callback=None*)

Creates an array of values by running each element in the collection through the callback. The callback is invoked with three arguments: `(value, index|key, collection)`. If a property name is passed for callback, the created *pluck()* style callback will return the property value of the given element. If an object is

passed for callback, the created *where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Mapped list.

Return type list

Example

```
>>> map_([1, 2, 3, 4], str)
['1', '2', '3', '4']
```

See also:

- *map_()* (main definition)
- *collect()* (alias)

New in version 1.0.0.

pydash.collections.**contains**(*collection*, *target*, *from_index=0*)

Checks if a given value is present in a collection. If *from_index* is negative, it is used as the offset from the end of the collection.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **target** (*mixed*) – Target value to compare to.
- **from_index** (*int, optional*) – Offset to start search from.

Returns Whether *target* is in *collection*.

Return type bool

Example

```
>>> contains([1, 2, 3, 4], 2)
True
>>> contains([1, 2, 3, 4], 2, from_index=2)
False
>>> contains({'a': 1, 'b': 2, 'c': 3, 'd': 4}, 2)
True
```

See also:

- *contains()* (main definition)
- *include()* (alias)

New in version 1.0.0.

pydash.collections.**count_by**(*collection*, *callback=None*)

Creates an object composed of keys generated from the results of running each element of *collection* through the callback.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Dict containing counts by key.

Return type dict

Example

```
>>> results = count_by([1, 2, 1, 2, 3, 4])
>>> assert results == {1: 2, 2: 2, 3: 1, 4: 1}
>>> results = count_by(['a', 'A', 'B', 'b'], lambda x: x.lower())
>>> assert results == {'a': 2, 'b': 2}
>>> results = count_by({'a': 1, 'b': 1, 'c': 3, 'd': 3})
>>> assert results == {1: 2, 3: 2}
```

New in version 1.0.0.

pydash.collections.**deep_pluck**(*collection*, *path*)

Like pluck but works with deep paths.

Parameters

- **collection** (*list|dict*) – list of dicts
- **path** (*str|list*) – collection's path to pluck

Returns plucked list

Return type list

Example

```
>>> deep_pluck([[[0, 1]], [[2, 3]], [[4, 5]]], '0.1')
[1, 3, 5]
>>> deep_pluck([{'a': {'b': 1}}, {'a': {'b': 2}}], 'a.b')
[1, 2]
>>> deep_pluck([{'a': {'b': [0, 1]}}, {'a': {'b': [2, 3]}}], 'a.b.1')
[1, 3]
```

New in version 2.2.0.

pydash.collections.**detect**(*collection*, *callback=None*)

Iterates over elements of a collection, returning the first element that the callback returns truthy for.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns First element found or `None`.

Return type mixed

Example

```
>>> find([1, 2, 3, 4], lambda x: x >= 3)
3
>>> find([{'a': 1}, {'b': 2}, {'a': 1, 'b': 2}], {'a': 1})
{'a': 1}
```

See also:

- *find()* (main definition)
- *detect()* (alias)
- *find_where()* (alias)

New in version 1.0.0.

pydash.collections.**each**(*collection*, *callback=None*)

Iterates over elements of a collection, executing the callback for each element.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns *collection*

Return type list|dict

Example

```
>>> results = {}
>>> def cb(x): results[x] = x ** 2
>>> each([1, 2, 3, 4], cb)
[1, 2, 3, 4]
>>> assert results == {1: 1, 2: 4, 3: 9, 4: 16}
```

See also:

- *for_each()* (main definition)
- *each()* (alias)

New in version 1.0.0.

pydash.collections.**each_right**(*collection*, *callback*)

This method is like *for_each()* except that it iterates over elements of a *collection* from right to left.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns *collection*

Return type list|dict

Example

```
>>> results = {'total': 1}
>>> def cb(x): results['total'] = x * results['total']
>>> each_right([1, 2, 3, 4], cb)
[1, 2, 3, 4]
>>> assert results == {'total': 24}
```

See also:

- *for_each_right()* (main definition)
- *each_right()* (alias)

New in version 1.0.0.

pydash.collections.**every**(*collection*, *callback=None*)

Checks if the callback returns a truthy value for all elements of a collection. The callback is invoked with three arguments: `(value, index|key, collection)`. If a property name is passed for callback, the created *pluck()* style callback will return the property value of the given element. If an object is passed for callback, the created *where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Whether all elements are truthy.

Return type bool

Example

```
>>> every([1, True, 'hello'])
True
>>> every([1, False, 'hello'])
False
>>> every([{'a': 1}, {'a': True}, {'a': 'hello'}], 'a')
True
>>> every([{'a': 1}, {'a': False}, {'a': 'hello'}], 'a')
False
>>> every([{'a': 1}, {'a': 1}], {'a': 1})
True
>>> every([{'a': 1}, {'a': 2}], {'a': 1})
False
```

See also:

- *every()* (main definition)
- *all_()* (alias)

New in version 1.0.0.

pydash.collections.**filter_**(*collection*, *callback=None*)

Iterates over elements of a collection, returning a list of all elements the callback returns truthy for.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Filtered list.

Return type list

Example

```
>>> results = filter_([{'a': 1}, {'b': 2}, {'a': 1, 'b': 3}], {'a': 1})
>>> assert results == [{'a': 1}, {'a': 1, 'b': 3}]
>>> filter_([1, 2, 3, 4], lambda x: x >= 3)
[3, 4]
```

See also:

- *select()* (main definition)
- *filter_()* (alias)

New in version 1.0.0.

pydash.collections.**find**(*collection*, *callback=None*)

Iterates over elements of a collection, returning the first element that the callback returns truthy for.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns First element found or `None`.

Return type mixed

Example

```
>>> find([1, 2, 3, 4], lambda x: x >= 3)
3
>>> find([{'a': 1}, {'b': 2}, {'a': 1, 'b': 2}], {'a': 1})
{'a': 1}
```

See also:

- *find()* (main definition)
- *detect()* (alias)
- *find_where()* (alias)

New in version 1.0.0.

pydash.collections.**find_last**(*collection*, *callback=None*)

This method is like *find()* except that it iterates over elements of a *collection* from right to left.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.

- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Last element found or `None`.

Return type mixed

Example

```
>>> find_last([1, 2, 3, 4], lambda x: x >= 3)
4
>>> results = find_last([{'a': 1}, {'b': 2}, {'a': 1, 'b': 2}],
>>> assert results == {'a': 1, 'b': 2}
```

New in version 1.0.0.

`pydash.collections.`**find_where**(*collection*, *callback=None*)

Iterates over elements of a collection, returning the first element that the callback returns truthy for.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns First element found or `None`.

Return type mixed

Example

```
>>> find([1, 2, 3, 4], lambda x: x >= 3)
3
>>> find([{'a': 1}, {'b': 2}, {'a': 1, 'b': 2}], {'a': 1})
{'a': 1}
```

See also:

- *find()* (main definition)
- *detect()* (alias)
- *find_where()* (alias)

New in version 1.0.0.

`pydash.collections.`**foldl**(*collection*, *callback=None*, *accumulator=None*)

Reduces a collection to a value which is the accumulated result of running each element in the collection through the callback, where each successive callback execution consumes the return value of the previous execution.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed*) – Callback applied per iteration.
- **accumulator** (*mixed, optional*) – Initial value of aggregator. Default is to use the result of the first iteration.

Returns Accumulator object containing results of reduction.

Return type mixed

Example

```
>>> reduce_([1, 2, 3, 4], lambda total, x: total * x)
24
```

See also:

- *reduce_()* (main definition)
- *foldl()* (alias)
- *inject()* (alias)

New in version 1.0.0.

pydash.collections.**foldr**(*collection*, *callback=None*, *accumulator=None*)

This method is like *reduce_()* except that it iterates over elements of a *collection* from right to left.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed*) – Callback applied per iteration.
- **accumulator** (*mixed, optional*) – Initial value of aggregator. Default is to use the result of the first iteration.

Returns Accumulator object containing results of reduction.

Return type mixed

Example

```
>>> reduce_right([1, 2, 3, 4], lambda total, x: total ** x)
4096
```

See also:

- *reduce_right()* (main definition)
- *foldr()* (alias)

New in version 1.0.0.

Changed in version 3.2.1: Fix bug where collection was not reversed correctly.

pydash.collections.**for_each**(*collection*, *callback=None*)

Iterates over elements of a collection, executing the callback for each element.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns *collection*

Return type list|dict

Example

```
>>> results = {}
>>> def cb(x): results[x] = x ** 2
>>> each([1, 2, 3, 4], cb)
[1, 2, 3, 4]
>>> assert results == {1: 1, 2: 4, 3: 9, 4: 16}
```

See also:

- *for_each()* (main definition)
- *each()* (alias)

New in version 1.0.0.

pydash.collections.**for_each_right**(*collection*, *callback*)

This method is like *for_each()* except that it iterates over elements of a *collection* from right to left.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns *collection*

Return type list|dict

Example

```
>>> results = {'total': 1}
>>> def cb(x): results['total'] = x * results['total']
>>> each_right([1, 2, 3, 4], cb)
[1, 2, 3, 4]
>>> assert results == {'total': 24}
```

See also:

- *for_each_right()* (main definition)
- *each_right()* (alias)

New in version 1.0.0.

pydash.collections.**group_by**(*collection*, *callback=None*)

Creates an object composed of keys generated from the results of running each element of a *collection* through the callback.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Results of grouping by *callback*.

Return type dict

Example

```
>>> results = group_by([{'a': 1, 'b': 2}, {'a': 3, 'b': 4}], 'a')
>>> assert results == {1: [{'a': 1, 'b': 2}], 3: [{'a': 3, 'b': 4}]}
>>> results = group_by([{'a': 1, 'b': 2}, {'a': 3, 'b': 4}], {'a': 1})
>>> assert results == {False: [{'a': 3, 'b': 4}],                                    True: [{'a': 1,
```

New in version 1.0.0.

pydash.collections.**include**(*collection*, *target*, *from_index=0*)

Checks if a given value is present in a collection. If *from_index* is negative, it is used as the offset from the end of the collection.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **target** (*mixed*) – Target value to compare to.
- **from_index** (*int, optional*) – Offset to start search from.

Returns Whether *target* is in *collection*.

Return type bool

Example

```
>>> contains([1, 2, 3, 4], 2)
True
>>> contains([1, 2, 3, 4], 2, from_index=2)
False
>>> contains({'a': 1, 'b': 2, 'c': 3, 'd': 4}, 2)
True
```

See also:

- *contains()* (main definition)
- *include()* (alias)

New in version 1.0.0.

pydash.collections.**index_by**(*collection*, *callback=None*)

Creates an object composed of keys generated from the results of running each element of the collection through the given callback.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Results of indexing by *callback*.

Return type dict

Example

```
>>> results = index_by([{'a': 1, 'b': 2}, {'a': 3, 'b': 4}], 'a')
>>> assert results == {1: {'a': 1, 'b': 2}, 3: {'a': 3, 'b': 4}}
```

New in version 1.0.0.

pydash.collections.**inject**(*collection*, *callback=None*, *accumulator=None*)

Reduces a collection to a value which is the accumulated result of running each element in the collection through the callback, where each successive callback execution consumes the return value of the previous execution.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed*) – Callback applied per iteration.
- **accumulator** (*mixed, optional*) – Initial value of aggregator. Default is to use the result of the first iteration.

Returns Accumulator object containing results of reduction.

Return type mixed

Example

```
>>> reduce_([1, 2, 3, 4], lambda total, x: total * x)
24
```

See also:

- *reduce_()* (main definition)
- *foldl()* (alias)
- *inject()* (alias)

New in version 1.0.0.

pydash.collections.**invoke**(*collection*, *method_name*, **args*, ***kargs*)

Invokes the method named by *method_name* on each element in the *collection* returning a list of the results of each invoked method.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **method_name** (*str*) – Name of method to invoke.
- **args** (*optional*) – Arguments to pass to method call.
- **kargs** (*optional*) – Keyword arguments to pass to method call.

Returns List of results of invoking method of each item.

Return type list

Example

```
>>> items = [[1, 2], [2, 3], [3, 4]]
>>> invoke(items, 'pop')
[2, 3, 4]
>>> items
[[1], [2], [3]]
>>> items = [[1, 2], [2, 3], [3, 4]]
>>> invoke(items, 'pop', 0)
[1, 2, 3]
>>> items
[[2], [3], [4]]
```

New in version 1.0.0.

pydash.collections.**map_**(*collection*, *callback=None*)

Creates an array of values by running each element in the collection through the callback. The callback is invoked with three arguments: `(value, index|key, collection)`. If a property name is passed for callback, the created *pluck()* style callback will return the property value of the given element. If an object is passed for callback, the created *where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Mapped list.

Return type list

Example

```
>>> map_([1, 2, 3, 4], str)
['1', '2', '3', '4']
```

See also:

- *map_()* (main definition)
- *collect()* (alias)

New in version 1.0.0.

pydash.collections.**mapiter**(*collection*, *callback=None*)

Like *map_()* except returns a generator.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Each mapped item.

Return type generator

Example

```
>>> gen = mapiter([1, 2, 3, 4], str)
>>> next(gen)
'1'
>>> next(gen)
'2'
>>> list(gen)
['3', '4']
```

New in version 2.1.0.

pydash.collections.**max_**(*collection*, *callback=None*, *default=<pydash.helpers._NoValue object>*)
Retrieves the maximum value of a *collection*.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.
- **default** – default value when collection is empty

Returns Maximum value.

Return type mixed

Example

```
>>> max_([1, 2, 3, 4])
4
>>> max_([{'a': 1}, {'a': 2}, {'a': 3}], 'a')
{'a': 3}
>>> max_([], default=-1)
-1
```

New in version 1.0.0.

pydash.collections.**min_**(*collection*, *callback=None*, *default=<pydash.helpers._NoValue object>*)
Retrieves the minimum value of a *collection*.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Minimum value.

Return type mixed

Example

```
>>> min_([1, 2, 3, 4])
1
>>> min_([{'a': 1}, {'a': 2}, {'a': 3}], 'a')
{'a': 1}
>>> min_([], default=100)
100
```

New in version 1.0.0.

pydash.collections.**partition**(*collection*, *callback=None*)

Creates an array of elements split into two groups, the first of which contains elements the *callback* returns truthy for, while the second of which contains elements the *callback* returns falsey for. The *callback* is invoked with three arguments: `(value, index|key, collection)`.

If a property name is provided for *callback* the created *pluck()* style callback returns the property value of the given element.

If an object is provided for *callback* the created *where()* style callback returns `True` for elements that have the properties of the given object, else `False`.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns List of grouped elements.

Return type list

Example

```
>>> partition([1, 2, 3, 4], lambda x: x >= 3)
[[3, 4], [1, 2]]
```

New in version 1.1.0.

pydash.collections.**pluck**(*collection*, *key*)

Retrieves the value of a specified property from all elements in the collection.

Parameters

- **collection** (*list*) – List of dicts.
- **key** (*str*) – Collection's key to pluck.

Returns Plucked list.

Return type list

Example

```
>>> pluck([{'a': 1, 'b': 2}, {'a': 3, 'b': 4}, {'a': 5, 'b': 6}], 'a')
[1, 3, 5]
```

New in version 1.0.0.

pydash.collections.**reduce_**(*collection*, *callback=None*, *accumulator=None*)

Reduces a collection to a value which is the accumulated result of running each element in the collection through the callback, where each successive callback execution consumes the return value of the previous execution.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed*) – Callback applied per iteration.
- **accumulator** (*mixed, optional*) – Initial value of aggregator. Default is to use the result of the first iteration.

Returns Accumulator object containing results of reduction.

Return type mixed

Example

```
>>> reduce_([1, 2, 3, 4], lambda total, x: total * x)
24
```

See also:

- *reduce_()* (main definition)
- *foldl()* (alias)
- *inject()* (alias)

New in version 1.0.0.

pydash.collections.**reduce_right**(*collection*, *callback=None*, *accumulator=None*)

This method is like *reduce_()* except that it iterates over elements of a *collection* from right to left.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed*) – Callback applied per iteration.
- **accumulator** (*mixed, optional*) – Initial value of aggregator. Default is to use the result of the first iteration.

Returns Accumulator object containing results of reduction.

Return type mixed

Example

```
>>> reduce_right([1, 2, 3, 4], lambda total, x: total ** x)
4096
```

See also:

- *reduce_right()* (main definition)
- *foldr()* (alias)

New in version 1.0.0.

Changed in version 3.2.1: Fix bug where collection was not reversed correctly.

pydash.collections.**reductions**(*collection*, *callback=None*, *accumulator=None*, *from_right=False*)

This function is like *reduce_()* except that it returns a list of every intermediate value in the reduction operation.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed*) – Callback applied per iteration.

- **accumulator** (*mixed, optional*) – Initial value of aggregator. Default is to use the result of the first iteration.

Returns Results of each reduction operation.

Return type list

Example

```
>>> reductions([1, 2, 3, 4], lambda total, x: total * x)
[2, 6, 24]
```

Note: The last element of the returned list would be the result of using *reduce_()*.

New in version 2.0.0.

pydash.collections.**reductions_right**(*collection*, *callback=None*, *accumulator=None*)

This method is like *reductions()* except that it iterates over elements of a *collection* from right to left.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed*) – Callback applied per iteration.
- **accumulator** (*mixed, optional*) – Initial value of aggregator. Default is to use the result of the first iteration.

Returns Results of each reduction operation.

Return type list

Example

```
>>> reductions_right([1, 2, 3, 4], lambda total, x: total ** x)
[64, 4096, 4096]
```

Note: The last element of the returned list would be the result of using *reduce_()*.

New in version 2.0.0.

pydash.collections.**reject**(*collection*, *callback=None*)

The opposite of *filter_()* this method returns the elements of a collection that the callback does **not** return truthy for.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Rejected elements of *collection*.

Return type list

Example

```
>>> reject([1, 2, 3, 4], lambda x: x >= 3)
[1, 2]
>>> reject([{'a': 0}, {'a': 1}, {'a': 2}], 'a')
[{'a': 0}]
>>> reject([{'a': 0}, {'a': 1}, {'a': 2}], {'a': 1})
[{'a': 0}, {'a': 2}]
```

New in version 1.0.0.

pydash.collections.**sample**(*collection*, *n=None*)

Retrieves a random element or *n* random elements from a *collection*.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **n** (*int, optional*) – Number of random samples to return.

Returns List of sampled collection value if *n* is provided, else single value from collection if *n* is `None`.

Return type list|mixed

Example

```
>>> items = [1, 2, 3, 4, 5]
>>> results = sample(items, 2)
>>> assert len(results) == 2
>>> assert set(items).intersection(results) == set(results)
```

New in version 1.0.0.

pydash.collections.**select**(*collection*, *callback=None*)

Iterates over elements of a collection, returning a list of all elements the callback returns truthy for.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Filtered list.

Return type list

Example

```
>>> results = filter_([{'a': 1}, {'b': 2}, {'a': 1, 'b': 3}], {'a': 1})
>>> assert results == [{'a': 1}, {'a': 1, 'b': 3}]
>>> filter_([1, 2, 3, 4], lambda x: x >= 3)
[3, 4]
```

See also:

- *select()* (main definition)
- *filter_()* (alias)

New in version 1.0.0.

pydash.collections.**shuffle**(*collection*)

Creates a list of shuffled values, using a version of the Fisher-Yates shuffle.

Parameters **collection** (*list|dict*) – Collection to iterate over.

Returns Shuffled list of values.

Return type list

Example

```
>>> items = [1, 2, 3, 4]
>>> results = shuffle(items)
>>> assert len(results) == len(items)
>>> assert set(results) == set(items)
```

New in version 1.0.0.

pydash.collections.**size**(*collection*)

Gets the size of the *collection* by returning *len(collection)* for iterable objects.

Parameters **collection** (*list|dict*) – Collection to iterate over.

Returns Collection length.

Return type int

Example

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

New in version 1.0.0.

pydash.collections.**some**(*collection*, *callback=None*)

Checks if the callback returns a truthy value for any element of a collection. The callback is invoked with three arguments: `(value, index|key, collection)`. If a property name is passed for callback, the created *pluck()* style callback will return the property value of the given element. If an object is passed for callback, the created *where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callbacked** (*mixed, optional*) – Callback applied per iteration.

Returns Whether any of the elements are truthy.

Return type bool

Example

```
>>> some([False, True, 0])
True
>>> some([False, 0, None])
False
```

```
>>> some([1, 2, 3, 4], lambda x: x >= 3)
True
>>> some([1, 2, 3, 4], lambda x: x == 0)
False
```

See also:

- *some()* (main definition)
- *any_()* (alias)

New in version 1.0.0.

pydash.collections.**sort_by**(*collection*, *callback=None*, *reverse=False*)

Creates a list of elements, sorted in ascending order by the results of running each element in a *collection* through the callback.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **callback** (*mixed, optional*) – Callback applied per iteration.
- **reverse** (*bool, optional*) – Whether to reverse the sort. Defaults to `False`.

Returns Sorted list.

Return type list

Example

```
>>> sort_by({'a': 2, 'b': 3, 'c': 1})
[1, 2, 3]
>>> sort_by({'a': 2, 'b': 3, 'c': 1}, reverse=True)
[3, 2, 1]
>>> sort_by([{'a': 2}, {'a': 3}, {'a': 1}], 'a')
[{'a': 1}, {'a': 2}, {'a': 3}]
```

New in version 1.0.0.

pydash.collections.**sort_by_all**(*collection*, *keys*, *orders=None*, *reverse=False*)

This method is like *sort_by()* except that it sorts by key names instead of an iteratee function. Keys can be sorted in descending order by prepending a `"-"` to the key name (e.g. `"name"` would become `"-name"`) or by passing a list of boolean sort options via *orders* where `True` is ascending and `False` is descending.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **keys** (*list*) – List of keys to sort by. By default, keys will be sorted in ascending order. To sort a key in descending order, prepend a `"-"` to the key name. For example, to sort the key value for `"name"` in descending order, use `"-name"`.
- **orders** (*list, optional*) – List of boolean sort orders to apply for each key. `True` corresponds to ascending order while `False` is descending. Defaults to `None`.
- **reverse** (*bool, optional*) – Whether to reverse the sort. Defaults to `False`.

Returns Sorted list.

Return type list

Example

```
>>> items = [{'a': 2, 'b': 1}, {'a': 3, 'b': 2}, {'a': 1, 'b': 3}]
>>> results = sort_by_all(items, ['b', 'a'])
>>> assert results == [{'a': 2, 'b': 1},                                {'a': 3, 'b': 2},
>>> results = sort_by_all(items, ['a', 'b'])
>>> assert results == [{'a': 1, 'b': 3},                                {'a': 2, 'b': 1},
>>> results = sort_by_all(items, ['-a', 'b'])
>>> assert results == [{'a': 3, 'b': 2},                                {'a': 2, 'b': 1},
>>> results = sort_by_all(items, ['a', 'b'], [False, True])
>>> assert results == [{'a': 3, 'b': 2},                                {'a': 2, 'b': 1},
```

See also:

- *sort_by_all()* (main definition)
- *sort_by_order()* (alias)

New in version 3.0.0.

Changed in version 3.2.0: Added *orders* argument.

Changed in version 3.2.0: Added *sort_by_order()* as alias.

pydash.collections.**sort_by_order**(*collection*, *keys*, *orders=None*, *reverse=False*)

This method is like *sort_by()* except that it sorts by key names instead of an iteratee function. Keys can be sorted in descending order by prepending a `"-"` to the key name (e.g. `"name"` would become `"-name"`) or by passing a list of boolean sort options via *orders* where `True` is ascending and `False` is descending.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **keys** (*list*) – List of keys to sort by. By default, keys will be sorted in ascending order. To sort a key in descending order, prepend a `"-"` to the key name. For example, to sort the key value for `"name"` in descending order, use `"-name"`.
- **orders** (*list, optional*) – List of boolean sort orders to apply for each key. `True` corresponds to ascending order while `False` is descending. Defaults to `None`.
- **reverse** (*bool, optional*) – Whether to reverse the sort. Defaults to `False`.

Returns Sorted list.

Return type list

Example

```
>>> items = [{'a': 2, 'b': 1}, {'a': 3, 'b': 2}, {'a': 1, 'b': 3}]
>>> results = sort_by_all(items, ['b', 'a'])
>>> assert results == [{'a': 2, 'b': 1},                                {'a': 3, 'b': 2},
>>> results = sort_by_all(items, ['a', 'b'])
>>> assert results == [{'a': 1, 'b': 3},                                {'a': 2, 'b': 1},
>>> results = sort_by_all(items, ['-a', 'b'])
>>> assert results == [{'a': 3, 'b': 2},                                {'a': 2, 'b': 1},
>>> results = sort_by_all(items, ['a', 'b'], [False, True])
>>> assert results == [{'a': 3, 'b': 2},                                {'a': 2, 'b': 1},
```

See also:

- *sort_by_all()* (main definition)
- *sort_by_order()* (alias)

New in version 3.0.0.

Changed in version 3.2.0: Added *orders* argument.

Changed in version 3.2.0: Added *sort_by_order()* as alias.

pydash.collections.**to_list**(*collection*)

Converts the collection to a list.

Parameters **collection** (*list|dict*) – Collection to iterate over.

Returns Collection converted to list.

Return type list

Example

```
>>> results = to_list({'a': 1, 'b': 2, 'c': 3})
>>> assert set(results) == set([1, 2, 3])
>>> to_list((1, 2, 3, 4))
[1, 2, 3, 4]
```

New in version 1.0.0.

pydash.collections.**where**(*collection*, *properties*)

Examines each element in a collection, returning an array of all elements that have the given properties.

Parameters

- **collection** (*list|dict*) – Collection to iterate over.
- **properties** (*dict*) – property values to filter by

Returns filtered list.

Return type list

Example

```
>>> results = where([{'a': 1}, {'b': 2}, {'a': 1, 'b': 3}], {'a': 1})
>>> assert results == [{'a': 1}, {'a': 1, 'b': 3}]
```

New in version 1.0.0.

4.1.5 Functions

Functions that wrap other functions.

New in version 1.0.0.

pydash.functions.**after**(*func*, *n*)

Creates a function that executes *func*, with the arguments of the created function, only after being called *n* times.

Parameters

- **func** (*function*) – Function to execute.

- **n** (*int*) – Number of times *func* must be called before it is executed.

Returns Function wrapped in an `After` context.

Return type After

Example

```
>>> func = lambda a, b, c: (a, b, c)
>>> after_func = after(func, 3)
>>> after_func(1, 2, 3)
>>> after_func(1, 2, 3)
>>> after_func(1, 2, 3)
(1, 2, 3)
>>> after_func(4, 5, 6)
(4, 5, 6)
```

New in version 1.0.0.

Changed in version 3.0.0: Reordered arguments to make *func* first.

`pydash.functions.`**`ary`**(*func*, *n*)

Creates a function that accepts up to *n* arguments ignoring any additional arguments. Only positional arguments are capped. All keyword arguments are allowed through.

Parameters

- **func** (*function*) – Function to cap arguments for.
- **n** (*int*) – Number of arguments to accept.

Returns Function wrapped in an `Ary` context.

Return type Ary

Example

```
>>> func = lambda a, b, c=0, d=5: (a, b, c, d)
>>> ary_func = ary(func, 2)
>>> ary_func(1, 2, 3, 4, 5, 6)
(1, 2, 0, 5)
>>> ary_func(1, 2, 3, 4, 5, 6, c=10, d=20)
(1, 2, 10, 20)
```

New in version 3.0.0.

`pydash.functions.`**`before`**(*func*, *n*)

Creates a function that executes *func*, with the arguments of the created function, until it has been called *n* times.

Parameters

- **func** (*function*) – Function to execute.
- **n** (*int*) – Number of times *func* may be executed.

Returns Function wrapped in an `Before` context.

Return type Before

Example

```
>>> func = lambda a, b, c: (a, b, c)
>>> before_func = before(func, 3)
>>> before_func(1, 2, 3)
(1, 2, 3)
>>> before_func(1, 2, 3)
(1, 2, 3)
>>> before_func(1, 2, 3)
>>> before_func(1, 2, 3)
```

New in version 1.1.0.

Changed in version 3.0.0: Reordered arguments to make *func* first.

pydash.functions.**compose**(**funcs*)

This function is like *flow()* except that it creates a function that invokes the provided functions from right to left. For example, composing the functions `f()`, `g()`, and `h()` produces `f(g(h()))`.

Parameters ***funcs** (*function*) – Function(s) to compose.

Returns Function(s) wrapped in a `Compose` context.

Return type Compose

Example

```
>>> mult_5 = lambda x: x * 5
>>> div_10 = lambda x: x / 10.0
>>> pow_2 = lambda x: x ** 2
>>> ops = flow_right(mult_5, div_10, pow_2, sum)
>>> ops([1, 2, 3, 4])
50.0
```

See also:

- *flow_right()* (main definition)
- *compose()* (alias)
- *pipe_right()* (alias)

New in version 1.0.0.

Changed in version 2.0.0: Added *flow_right()* and made *compose()* an alias.

Changed in version 2.3.1: Added *pipe_right()* as alias.

pydash.functions.**conjoin**(**funcs*)

Creates a function that composes multiple predicate functions into a single predicate that tests whether **all** elements of an object pass each predicate.

Parameters ***funcs** (*function*) – Function(s) to conjoin.

Returns Function(s) wrapped in a `Conjoin` context.

Return type Conjoin

Example

```
>>> conjoiner = conjoin(lambda x: isinstance(x, int), lambda x: x > 3)
>>> conjoiner([1, 2, 3])
False
>>> conjoiner([1.0, 2, 1])
False
>>> conjoiner([4.0, 5, 6])
False
>>> conjoiner([4, 5, 6])
True
```

New in version 2.0.0.

pydash.functions.**curry**(*func*, *arity=None*)

Creates a function that accepts one or more arguments of *func* that when invoked either executes *func* returning its result (if all *func* arguments have been provided) or returns a function that accepts one or more of the remaining *func* arguments, and so on.

Parameters

- **func** (*function*) – Function to curry.
- **arity** (*int, optional*) – Number of function arguments that can be accepted by curried function. Default is to use the number of arguments that are accepted by *func*.

Returns Function wrapped in a `Curry` context.

Return type Curry

Example

```
>>> func = lambda a, b, c: (a, b, c)
>>> currier = curry(func)
>>> currier = currier(1)
>>> assert isinstance(currier, Curry)
>>> currier = currier(2)
>>> assert isinstance(currier, Curry)
>>> currier = currier(3)
>>> currier
(1, 2, 3)
```

New in version 1.0.0.

pydash.functions.**curry_right**(*func*, *arity=None*)

This method is like *curry()* except that arguments are applied to *func* in the manner of *partial_right()* instead of *partial()*.

Parameters

- **func** (*function*) – Function to curry.
- **arity** (*int, optional*) – Number of function arguments that can be accepted by curried function. Default is to use the number of arguments that are accepted by *func*.

Returns Function wrapped in a `CurryRight` context.

Return type CurryRight

Example

```
>>> func = lambda a, b, c: (a, b, c)
>>> currier = curry_right(func)
>>> currier = currier(1)
>>> assert isinstance(currier, CurryRight)
>>> currier = currier(2)
>>> assert isinstance(currier, CurryRight)
>>> currier = currier(3)
>>> currier
(3, 2, 1)
```

New in version 1.1.0.

pydash.functions.**debounce**(*func*, *wait*, *max_wait=False*)

Creates a function that will delay the execution of *func* until after *wait* milliseconds have elapsed since the last time it was invoked. Subsequent calls to the debounced function will return the result of the last *func* call.

Parameters

- **func** (*function*) – Function to execute.
- **wait** (*int*) – Milliseconds to wait before executing *func*.
- **max_wait** (*optional*) – Maximum time to wait before executing *func*.

Returns Function wrapped in a `Debounce` context.

Return type Debounce

New in version 1.0.0.

pydash.functions.**delay**(*func*, *wait*, **args*, ***kargs*)

Executes the *func* function after *wait* milliseconds. Additional arguments will be provided to *func* when it is invoked.

Parameters

- **func** (*function*) – Function to execute.
- **wait** (*int*) – Milliseconds to wait before executing *func*.
- ***args** (*optional*) – Arguments to pass to *func*.
- ****kargs** (*optional*) – Keyword arguments to pass to *func*.

Returns Return from *func*.

Return type mixed

New in version 1.0.0.

pydash.functions.**disjoin**(**funcs*)

Creates a function that composes multiple predicate functions into a single predicate that tests whether **any** elements of an object pass each predicate.

Parameters ***funcs** (*function*) – Function(s) to disjoin.

Returns Function(s) wrapped in a `Disjoin` context.

Return type Disjoin

Example

```
>>> disjoiner = disjoin(lambda x: isinstance(x, float),
>>> disjoiner([1, '2', '3'])
True
>>> disjoiner([1.0, '2', '3'])
True
>>> disjoiner(['1', '2', '3'])
False
```

lambda x:

New in version 2.0.0.

pydash.functions.**flow**(**funcs*)

Creates a function that is the composition of the provided functions, where each successive invocation is supplied the return value of the previous. For example, composing the functions `f()`, `g()`, and `h()` produces `h(g(f()))`.

Parameters ***funcs** (*function*) – Function(s) to compose.

Returns Function(s) wrapped in a `Compose` context.

Return type Compose

Example

```
>>> mult_5 = lambda x: x * 5
>>> div_10 = lambda x: x / 10.0
>>> pow_2 = lambda x: x ** 2
>>> ops = flow(sum, mult_5, div_10, pow_2)
>>> ops([1, 2, 3, 4])
25.0
```

See also:

- *flow()* (main definition)
- *pipe()* (alias)

New in version 2.0.0.

Changed in version 2.3.1: Added *pipe()* as alias.

pydash.functions.**flow_right**(**funcs*)

This function is like *flow()* except that it creates a function that invokes the provided functions from right to left. For example, composing the functions `f()`, `g()`, and `h()` produces `f(g(h()))`.

Parameters ***funcs** (*function*) – Function(s) to compose.

Returns Function(s) wrapped in a `Compose` context.

Return type Compose

Example

```
>>> mult_5 = lambda x: x * 5
>>> div_10 = lambda x: x / 10.0
>>> pow_2 = lambda x: x ** 2
>>> ops = flow_right(mult_5, div_10, pow_2, sum)
```

```
>>> ops([1, 2, 3, 4])
50.0
```

See also:

- *flow_right()* (main definition)
- *compose()* (alias)
- *pipe_right()* (alias)

New in version 1.0.0.

Changed in version 2.0.0: Added *flow_right()* and made *compose()* an alias.

Changed in version 2.3.1: Added *pipe_right()* as alias.

pydash.functions.**iterated**(*func*)

Creates a function that is composed with itself. Each call to the iterated function uses the previous function call's result as input. Returned `Iterated` instance can be called with `(initial, n)` where *initial* is the initial value to seed *func* with and *n* is the number of times to call *func*.

Parameters **func** (*function*) – Function to iterate.

Returns Function wrapped in a `Iterated` context.

Return type Iterated

Example

```
>>> doubler = iterated(lambda x: x * 2)
>>> doubler(4, 5)
128
>>> doubler(3, 9)
1536
```

New in version 2.0.0.

pydash.functions.**juxtapose**(**funcs*)

Creates a function whose return value is a list of the results of calling each *funcs* with the supplied arguments.

Parameters ***funcs** (*function*) – Function(s) to juxtapose.

Returns Function wrapped in a `Juxtapose` context.

Return type Juxtapose

Example

```
>>> double = lambda x: x * 2
>>> triple = lambda x: x * 3
>>> quadruple = lambda x: x * 4
>>> juxtapose(double, triple, quadruple)(5)
[10, 15, 20]
```

New in version 2.0.0.

pydash.functions.**mod_args**(*func*, **transforms*)

Creates a function that runs each argument through a corresponding transform function.

Parameters

- **func** (*function*) – Function to wrap.
- ***transforms** (*function*) – Functions to transform arguments, specified as individual functions or lists of functions.

Returns Function wrapped in a `ModArgs` context.

Return type ModArgs

Example

```
>>> squared = lambda x: x ** 2
>>> double = lambda x: x * 2
>>> modder = mod_args(lambda x, y: [x, y], squared, double)
>>> modder(5, 10)
[25, 20]
```

New in version 3.3.0.

`pydash.functions.`**`negate`**(*func*)

Creates a function that negates the result of the predicate *func*. The *func* function is executed with the arguments of the created function.

Parameters **func** (*function*) – Function to negate execute.

Returns Function wrapped in a `Negate` context.

Return type Negate

Example

```
>>> not_is_number = negate(lambda x: isinstance(x, (int, float)))
>>> not_is_number(1)
False
>>> not_is_number('1')
True
```

New in version 1.1.0.

`pydash.functions.`**`once`**(*func*)

Creates a function that is restricted to execute *func* once. Repeat calls to the function will return the value of the first call.

Parameters **func** (*function*) – Function to execute.

Returns Function wrapped in a `Once` context.

Return type Once

Example

```
>>> oncer = once(lambda *args: args[0])
>>> oncer(5)
5
>>> oncer(6)
5
```

New in version 1.0.0.

pydash.functions.**partial**(*func*, **args*, ***kargs*)

Creates a function that, when called, invokes *func* with any additional partial arguments prepended to those provided to the new function.

Parameters

- **func** (*function*) – Function to execute.
- ***args** (*optional*) – Partial arguments to prepend to function call.
- ****kargs** (*optional*) – Partial keyword arguments to bind to function call.

Returns Function wrapped in a `Partial` context.

Return type Partial

Example

```
>>> dropper = partial(lambda array, n: array[n:], [1, 2, 3, 4])
>>> dropper(2)
[3, 4]
>>> dropper(1)
[2, 3, 4]
>>> myrest = partial(lambda array, n: array[n:], n=1)
>>> myrest([1, 2, 3, 4])
[2, 3, 4]
```

New in version 1.0.0.

pydash.functions.**partial_right**(*func*, **args*, ***kargs*)

This method is like *partial()* except that partial arguments are appended to those provided to the new function.

Parameters

- **func** (*function*) – Function to execute.
- ***args** (*optional*) – Partial arguments to append to function call.
- ****kargs** (*optional*) – Partial keyword arguments to bind to function call.

Returns Function wrapped in a `Partial` context.

Return type Partial

Example

```
>>> myrest = partial_right(lambda array, n: array[n:], 1)
>>> myrest([1, 2, 3, 4])
[2, 3, 4]
```

New in version 1.0.0.

pydash.functions.**pipe**(**funcs*)

Creates a function that is the composition of the provided functions, where each successive invocation is supplied the return value of the previous. For example, composing the functions `f()`, `g()`, and `h()` produces `h(g(f()))`.

Parameters ***funcs** (*function*) – Function(s) to compose.

Returns Function(s) wrapped in a `Compose` context.

Return type Compose

Example

```
>>> mult_5 = lambda x: x * 5
>>> div_10 = lambda x: x / 10.0
>>> pow_2 = lambda x: x ** 2
>>> ops = flow(sum, mult_5, div_10, pow_2)
>>> ops([1, 2, 3, 4])
25.0
```

See also:

- *`flow()`* (main definition)
- *`pipe()`* (alias)

New in version 2.0.0.

Changed in version 2.3.1: Added *`pipe()`* as alias.

`pydash.functions.`**`pipe_right`**`(*funcs)`

This function is like *`flow()`* except that it creates a function that invokes the provided functions from right to left. For example, composing the functions `f()`, `g()`, and `h()` produces `f(g(h()))`.

Parameters **`*funcs`** (*`function`*) – Function(s) to compose.

Returns Function(s) wrapped in a `Compose` context.

Return type Compose

Example

```
>>> mult_5 = lambda x: x * 5
>>> div_10 = lambda x: x / 10.0
>>> pow_2 = lambda x: x ** 2
>>> ops = flow_right(mult_5, div_10, pow_2, sum)
>>> ops([1, 2, 3, 4])
50.0
```

See also:

- *`flow_right()`* (main definition)
- *`compose()`* (alias)
- *`pipe_right()`* (alias)

New in version 1.0.0.

Changed in version 2.0.0: Added *`flow_right()`* and made *`compose()`* an alias.

Changed in version 2.3.1: Added *`pipe_right()`* as alias.

`pydash.functions.`**`rearg`**`(func, *indexes)`

Creates a function that invokes *func* with arguments arranged according to the specified indexes where the argument value at the first index is provided as the first argument, the argument value at the second index is provided as the second argument, and so on.

Parameters

- **func** (*function*) – Function to rearrange arguments for.
- ***indexes** (*int*) – The arranged argument indexes.

Returns Function wrapped in a `Rearg` context.

Return type Rearg

Example

```
>>> jumble = rearg(lambda *args: args, 1, 2, 3)
>>> jumble(1, 2, 3)
(2, 3, 1)
>>> jumble('a', 'b', 'c', 'd', 'e')
('b', 'c', 'd', 'a', 'e')
```

New in version 3.0.0.

`pydash.functions.`**spread**(*func*)

Creates a function that invokes *func* with the array of arguments provided to the created function.

Parameters **func** (*function*) – Function to spread.

Returns Function wrapped in a `Spread` context.

Return type Spread

Example

```
>>> greet = spread(lambda people: 'Hello ' + ', '.join(people) + '!')
>>> greet(['Mike', 'Don', 'Leo'])
'Hello Mike, Don, Leo!'
```

New in version 3.1.0.

`pydash.functions.`**throttle**(*func*, *wait*)

Creates a function that, when executed, will only call the *func* function at most once per every *wait* milliseconds. Subsequent calls to the throttled function will return the result of the last *func* call.

Parameters

- **func** (*function*) – Function to throttle.
- **wait** (*int*) – Milliseconds to wait before calling *func* again.

Returns Results of last *func* call.

Return type mixed

New in version 1.0.0.

`pydash.functions.`**wrap**(*value*, *func*)

Creates a function that provides value to the wrapper function as its first argument. Additional arguments provided to the function are appended to those provided to the wrapper function.

Parameters

- **value** (*mixed*) – Value provided as first argument to function call.
- **func** (*function*) – Function to execute.

Returns Function wrapped in a `Partial` context.

Return type Partial

Example

```
>>> wrapper = wrap('hello', lambda *args: args)
>>> wrapper(1, 2)
('hello', 1, 2)
```

New in version 1.0.0.

4.1.6 Numerical

Numerical/mathemetical related functions.

New in version 2.1.0.

`pydash.numerical.`**`add`**(*collection*, *callback=None*)

Sum each element in *collection*. If callback is passed, each element of *collection* is passed through a callback before the summation is computed. If *collection* and *callback* are numbers, they are added together.

Parameters

- **`collection`** (*`list|dict|number`*) – Collection to process or first number to add.
- **`callback`** (*`mixed|number, optional`*) – Callback applied per iteration or second number to add.

Returns Result of summation.

Return type number

Example

```
>>> add([1, 2, 3, 4])
10
>>> add([1, 2, 3, 4], lambda x: x ** 2)
30
>>> add(1, 5)
6
```

See also:

- *`add()`* (main definition)
- *`sum_()`* (alias)

New in version 2.1.0.

Changed in version 3.3.0: Support adding two numbers when passed as positional arguments.

`pydash.numerical.`**`average`**(*collection*, *callback=None*)

Calculate arithmetic mean of each element in *collection*. If callback is passed, each element of *collection* is passed through a callback before the mean is computed.

Parameters

- **`collection`** (*`list|dict`*) – Collection to process.

- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Result of mean.

Return type float

Example

```
>>> average([1, 2, 3, 4])
2.5
>>> average([1, 2, 3, 4], lambda x: x ** 2)
7.5
```

See also:

- *average()* (main definition)
- *avg()* (alias)
- *mean()* (alias)

New in version 2.1.0.

pydash.numerical.**avg**(*collection*, *callback=None*)

Calculate arithmetic mean of each element in *collection*. If callback is passed, each element of *collection* is passed through a callback before the mean is computed.

Parameters

- **collection** (*list|dict*) – Collection to process.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Result of mean.

Return type float

Example

```
>>> average([1, 2, 3, 4])
2.5
>>> average([1, 2, 3, 4], lambda x: x ** 2)
7.5
```

See also:

- *average()* (main definition)
- *avg()* (alias)
- *mean()* (alias)

New in version 2.1.0.

pydash.numerical.**ceil**(*x*, *precision=0*)

Round number up to precision.

Parameters

- **x** (*number*) – Number to round up.

- **precision** (*int, optional*) – Rounding precision. Defaults to `0`.

Returns Number rounded up.

Return type int

Example

```
>>> ceil(3.275) == 4.0
True
>>> ceil(3.215, 1) == 3.3
True
>>> ceil(6.004, 2) == 6.01
True
```

New in version 3.3.0.

`pydash.numerical.`**`curve`**(*x*, *precision=0*)

Round number to precision.

Parameters

- **x** (*number*) – Number to round.
- **precision** (*int, optional*) – Rounding precision. Defaults to `0`.

Returns Rounded number.

Return type int

Example

```
>>> round_(3.275) == 3.0
True
>>> round_(3.275, 1) == 3.3
True
```

See also:

- *round_()* (main definition)
- *curve()* (alias)

New in version 2.1.0.

`pydash.numerical.`**`floor`**(*x*, *precision=0*)

Round number down to precision.

Parameters

- **x** (*number*) – Number to round down.
- **precision** (*int, optional*) – Rounding precision. Defaults to `0`.

Returns Number rounded down.

Return type int

Example

```
>>> floor(3.75) == 3.0
True
>>> floor(3.215, 1) == 3.2
True
>>> floor(0.046, 2) == 0.04
True
```

New in version 3.3.0.

pydash.numerical.**mean**(*collection*, *callback=None*)

Calculate arithmetic mean of each element in *collection*. If callback is passed, each element of *collection* is passed through a callback before the mean is computed.

Parameters

- **collection** (*list|dict*) – Collection to process.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Result of mean.

Return type float

Example

```
>>> average([1, 2, 3, 4])
2.5
>>> average([1, 2, 3, 4], lambda x: x ** 2)
7.5
```

See also:

- *average()* (main definition)
- *avg()* (alias)
- *mean()* (alias)

New in version 2.1.0.

pydash.numerical.**median**(*collection*, *callback=None*)

Calculate median of each element in *collection*. If callback is passed, each element of *collection* is passed through a callback before the median is computed.

Parameters

- **collection** (*list|dict*) – Collection to process.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Result of median.

Return type float

Example

```
>>> median([1, 2, 3, 4, 5])
3
>>> median([1, 2, 3, 4])
2.5
```

New in version 2.1.0.

pydash.numerical.**moving_average**(*array*, *size*)

Calculate moving average of each element of *array*. If callback is passed, each element of *array* is passed through a callback before the moving average is computed.

Parameters

- **array** (*list*) – List to process.
- **size** (*int*) – Window size.

Returns Result of moving average.

Return type list

Example

```
>>> moving_average(range(10), 1)
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
>>> moving_average(range(10), 5)
[2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
>>> moving_average(range(10), 10)
[4.5]
```

See also:

- moving_averge() (main definition)
- *moving_avg()* (alias)

New in version 2.1.0.

pydash.numerical.**moving_avg**(*array*, *size*)

Calculate moving average of each element of *array*. If callback is passed, each element of *array* is passed through a callback before the moving average is computed.

Parameters

- **array** (*list*) – List to process.
- **size** (*int*) – Window size.

Returns Result of moving average.

Return type list

Example

```
>>> moving_average(range(10), 1)
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
>>> moving_average(range(10), 5)
[2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
>>> moving_average(range(10), 10)
[4.5]
```

See also:

- `moving_averge()` (main definition)
- *moving_avg()* (alias)

New in version 2.1.0.

`pydash.numerical.pow_(x, n)`
Calculate exponentiation of *x* raised to the *n* power.

Parameters

- **x** (*number*) – Base number.
- **n** (*number*) – Exponent.

Returns Result of calculation.

Return type number

Example

```
>>> power(5, 2)
25
>>> power(12.5, 3)
1953.125
```

See also:

- *power()* (main definition)
- *pow_()* (alias)

New in version 2.1.0.

`pydash.numerical.power(x, n)`
Calculate exponentiation of *x* raised to the *n* power.

Parameters

- **x** (*number*) – Base number.
- **n** (*number*) – Exponent.

Returns Result of calculation.

Return type number

Example

```
>>> power(5, 2)
25
>>> power(12.5, 3)
1953.125
```

See also:

- *power()* (main definition)
- *pow_()* (alias)

New in version 2.1.0.

pydash.numerical.**round_**(*x*, *precision=0*)
Round number to precision.

Parameters

- **x** (*number*) – Number to round.
- **precision** (*int, optional*) – Rounding precision. Defaults to 0.

Returns Rounded number.

Return type int

Example

```
>>> round_(3.275) == 3.0
True
>>> round_(3.275, 1) == 3.3
True
```

See also:

- *round_()* (main definition)
- *curve()* (alias)

New in version 2.1.0.

pydash.numerical.**scale**(*array*, *maximum=1*)
Scale list of value to a maximum number.

Parameters

- **array** (*list*) – Numbers to scale.
- **maximum** (*number*) – Maximum scale value.

Returns Scaled numbers.

Return type list

Example

```
>>> scale([1, 2, 3, 4])
[0.25, 0.5, 0.75, 1.0]
>>> scale([1, 2, 3, 4], 1)
[0.25, 0.5, 0.75, 1.0]
>>> scale([1, 2, 3, 4], 4)
[1.0, 2.0, 3.0, 4.0]
>>> scale([1, 2, 3, 4], 2)
[0.5, 1.0, 1.5, 2.0]
```

New in version 2.1.0.

pydash.numerical.**sigma**(*array*)
Calculate standard deviation of list of numbers.

Parameters **array** (*list*) – List to process.

Returns Calculated standard deviation.

Return type float

Example

```
>>> round(std_deviation([1, 18, 20, 4]), 2) == 8.35
True
```

See also:

- *std_deviation()* (main definition)
- *sigma()* (alias)

New in version 2.1.0.

pydash.numerical.**slope**(*point1*, *point2*)
Calculate the slope between two points.

Parameters

- **point1** (*list|tuple*) – X and Y coordinates of first point.
- **point2** (*list|tuple*) – X and Y cooredinates of second point.

Returns Calculated slope.

Return type float

Example

```
>>> slope((1, 2), (4, 8))
2.0
```

New in version 2.1.0.

pydash.numerical.**std_deviation**(*array*)
Calculate standard deviation of list of numbers.

Parameters **array** (*list*) – List to process.

Returns Calculated standard deviation.

Return type float

Example

```
>>> round(std_deviation([1, 18, 20, 4]), 2) == 8.35
True
```

See also:

- *std_deviation()* (main definition)
- *sigma()* (alias)

New in version 2.1.0.

pydash.numerical.**sum_**(*collection*, *callback=None*)

Sum each element in *collection*. If callback is passed, each element of *collection* is passed through a callback before the summation is computed. If *collection* and *callback* are numbers, they are added together.

Parameters

- **collection** (*list|dict|number*) – Collection to process or first number to add.
- **callback** (*mixed|number, optional*) – Callback applied per iteration or second number to add.

Returns Result of summation.

Return type number

Example

```
>>> add([1, 2, 3, 4])
10
>>> add([1, 2, 3, 4], lambda x: x ** 2)
30
>>> add(1, 5)
6
```

See also:

- *add()* (main definition)
- *sum_()* (alias)

New in version 2.1.0.

Changed in version 3.3.0: Support adding two numbers when passed as positional arguments.

pydash.numerical.**transpose**(*array*)

Transpose the elements of *array*.

Parameters **array** (*list*) – List to process.

Returns Transposed list.

Return type list

Example

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

New in version 2.1.0.

pydash.numerical.**variance**(*array*)

Calculate the variance of the elements in *array*.

Parameters **array** (*list*) – List to process.

Returns Calculated variance.

Return type float

Example

```
>>> variance([1, 18, 20, 4])
69.6875
```

New in version 2.1.0.

`pydash.numerical.`**`zscore`**(*collection*, *callback=None*)

Calculate the standard score assuming normal distribution. If callback is passed, each element of *collection* is passed through a callback before the standard score is computed.

Parameters

- **`collection`** (*`list|dict`*) – Collection to process.
- **`callback`** (*`mixed, optional`*) – Callback applied per iteration.

Returns Calculated standard score.

Return type float

Example

```
>>> results = zscore([1, 2, 3])
```

[-1.224744871391589, 0.0, 1.224744871391589]

New in version 2.1.0.

4.1.7 Objects

Functions that operate on lists, dicts, and other objects.

New in version 1.0.0.

`pydash.objects.`**`assign`**(*obj*, **sources*, ***kargs*)

Assigns own enumerable properties of source object(s) to the destination object. If *callback* is supplied, it is invoked with two arguments: `(obj_value, source_value)`.

Parameters

- **`obj`** (*`dict`*) – Destination object whose properties will be modified.
- **`sources`** (*`dict`*) – Source objects to assign to *obj*.

Keyword Arguments **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Modified *obj*.

Return type dict

Warning: *obj* is modified in place.

Example

```
>>> obj = {}
>>> obj2 = assign(obj, {'a': 1}, {'b': 2}, {'c': 3})
>>> obj == {'a': 1, 'b': 2, 'c': 3}
True
>>> obj is obj2
True
```

See also:

- *assign()* (main definition)
- *extend()* (alias)

New in version 1.0.0.

Changed in version 2.3.2: Apply *clone_deep()* to each *source* before assigning to *obj*.

Changed in version 3.0.0: Allow callbacks to accept partial arguments.

Changed in version 3.4.4: Shallow copy each *source* instead of deep copying.

pydash.objects.**callables**(*obj*)

Creates a sorted list of keys of an object that are callable.

Parameters **obj** (*list|dict*) – Object to inspect.

Returns All keys whose values are callable.

Return type list

Example

```
>>> callables({'a': 1, 'b': lambda: 2, 'c': lambda: 3})
['b', 'c']
```

See also:

- *callables()* (main definition)
- *methods()* (alias)

New in version 1.0.0.

Changed in version 2.0.0: Renamed `functions` to `callables`.

pydash.objects.**clone**(*value*, *is_deep=False*, *callback=None*)

Creates a clone of *value*. If *is_deep* is `True` nested valueects will also be cloned, otherwise they will be assigned by reference. If a callback is provided it will be executed to produce the cloned values. The callback is invoked with one argument: `(value)`.

Parameters

- **value** (*list|dict*) – Object to clone.
- **is_deep** (*bool, optional*) – Whether to perform deep clone.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Example

```
>>> x = {'a': 1, 'b': 2, 'c': {'d': 3}}
>>> y = clone(x)
>>> y == y
True
>>> x is y
False
>>> x['c'] is y['c']
True
>>> z = clone(x, is_deep=True)
>>> x == z
True
>>> x['c'] is z['c']
False
```

Returns Cloned object.

Return type list|dict

New in version 1.0.0.

pydash.objects.**clone_deep**(*value*, *callback=None*)

Creates a deep clone of *value*. If a callback is provided it will be executed to produce the cloned values. The callback is invoked with one argument: `(value)`.

Parameters

- **value** (*list|dict*) – Object to clone.
- **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Cloned object.

Return type list|dict

Example

```
>>> x = {'a': 1, 'b': 2, 'c': {'d': 3}}
>>> y = clone_deep(x)
>>> y == y
True
>>> x is y
False
>>> x['c'] is y['c']
False
```

New in version 1.0.0.

pydash.objects.**deep_get**(*obj*, *path*, *default=None*)

Get the value at any depth of a nested object based on the path described by *path*. If path doesn't exist, *default* is returned.

Parameters

- **obj** (*list|dict*) – Object to process.
- **path** (*str|list*) – List or `.` delimited string of path describing path.

Keyword Arguments **default** (*mixed*) – Default value to return if path doesn't exist. Defaults to `None`.

Returns Value of *obj* at path.

Return type mixed

Example

```
>>> get({}, 'a.b.c') is None
True
>>> get({'a': {'b': {'c': [1, 2, 3, 4]}}}, 'a.b.c.1')
2
>>> get({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.1')
2
>>> get({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.2') is None
True
```

See also:

- *get()* (main definition)
- *get_path()* (alias)
- *deep_get()* (alias)

New in version 2.0.0.

Changed in version 2.2.0: Support escaping "." delimiter in single string path key.

Changed in version 3.3.0: Added *get()* as main definition and *get_path()* as alias. Made *deep_get()* an alias.

Changed in version 3.4.7: Fixed bug where an iterable default was iterated over instead of being returned when an object path wasn't found.

pydash.objects.**deep_has**(*obj*, *path*)

Checks if *path* exists as a key of *obj*.

Parameters

- **obj** (*mixed*) – Object to test.
- **path** (*mixed*) – Path to test for. Can be a list of nested keys or a `.` delimited string of path describing the path.

Returns Whether *obj* has *path*.

Return type bool

Example

```
>>> has([1, 2, 3], 1)
True
>>> has({'a': 1, 'b': 2}, 'b')
True
>>> has({'a': 1, 'b': 2}, 'c')
False
>>> has({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.1')
True
```

```
>>> has({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.2')
False
```

See also:

- *has()* (main definition)
- *deep_has()* (alias)
- *has_path()* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Return `False` on `ValueError` when checking path.

Changed in version 3.3.0: Added *deep_has()* as alias. Added *has_path()* as alias.

pydash.objects.**deep_set**(*obj*, *path*, *value*)

Sets the value of an object described by *path*. If any part of the object path doesn't exist, it will be created.

Parameters

- **obj** (*list|dict*) – Object to modify.
- **path** (*str | list*) – Target path to set value to.
- **value** (*mixed*) – Value to set.

Returns Modified *obj*.

Return type mixed

Example

```
>>> set_({}, 'a.b.c', 1)
{'a': {'b': {'c': 1}}}
>>> set_({}, 'a.0.c', 1)
{'a': {'0': {'c': 1}}}
>>> set_([1, 2], '2.0', 1)
[1, 2, [1]]
```

New in version 2.2.0.

Changed in version 3.3.0: Added *set_()* as main definition and *deep_set()* as alias.

pydash.objects.**deep_map_values**(*obj*, *callback=None*, *property_path=<pydash.helpers._NoValue object>*)

Map all non-object values in *obj* with return values from *callback*. The callback is invoked with two arguments: `(obj_value, property_path)` where `property_path` contains the list of path keys corresponding to the path of `obj_value`.

Parameters

- **obj** (*list|dict*) – Object to map.
- **callback** (*function*) – Callback applied to each value.

Returns The modified object.

Return type mixed

Warning: *obj* is modified in place.

Example

```
>>> x = {'a': 1, 'b': {'c': 2}}
>>> y = deep_map_values(x, lambda val: val * 2)
>>> y == {'a': 2, 'b': {'c': 4}}
True
>>> z = deep_map_values(x, lambda val, props: props)
>>> z == {'a': ['a'], 'b': {'c': ['b', 'c']}}
True
```

Changed in version 3.0.0: Allow callbacks to accept partial arguments.

pydash.objects.**defaults**(*obj*, **sources*)

Assigns own enumerable properties of source object(s) to the destination object for all destination properties that resolve to undefined.

Parameters

- **obj** (*dict*) – Destination object whose properties will be modified.
- **sources** (*dict*) – Source objects to assign to *obj*.

Returns Modified *obj*.

Return type dict

Warning: *obj* is modified in place.

Example

```
>>> obj = {'a': 1}
>>> obj2 = defaults(obj, {'b': 2}, {'c': 3}, {'a': 4})
>>> obj is obj2
True
>>> obj == {'a': 1, 'b': 2, 'c': 3}
True
```

New in version 1.0.0.

pydash.objects.**defaults_deep**(*obj*, **sources*)

This method is like *defaults()* except that it recursively assigns default properties.

Parameters

- **obj** (*dict*) – Destination object whose properties will be modified.
- **sources** (*dict*) – Source objects to assign to *obj*.

Returns Modified *obj*.

Return type dict

Warning: *obj* is modified in place.

Example

```
>>> obj = {'a': {'b': 1}}
>>> obj2 = defaults_deep(obj, {'a': {'b': 2, 'c': 3}})
>>> obj is obj2
True
>>> obj == {'a': {'b': 1, 'c': 3}}
True
```

New in version 3.3.0.

pydash.objects.**extend**(*obj*, **sources*, ***kargs*)

Assigns own enumerable properties of source object(s) to the destination object. If *callback* is supplied, it is invoked with two arguments: `(obj_value, source_value)`.

Parameters

- **obj** (*dict*) – Destination object whose properties will be modified.
- **sources** (*dict*) – Source objects to assign to *obj*.

Keyword Arguments **callback** (*mixed, optional*) – Callback applied per iteration.

Returns Modified *obj*.

Return type dict

Warning: *obj* is modified in place.

Example

```
>>> obj = {}
>>> obj2 = assign(obj, {'a': 1}, {'b': 2}, {'c': 3})
>>> obj == {'a': 1, 'b': 2, 'c': 3}
True
>>> obj is obj2
True
```

See also:

- *assign()* (main definition)
- *extend()* (alias)

New in version 1.0.0.

Changed in version 2.3.2: Apply *clone_deep()* to each *source* before assigning to *obj*.

Changed in version 3.0.0: Allow callbacks to accept partial arguments.

Changed in version 3.4.4: Shallow copy each *source* instead of deep copying.

pydash.objects.**find_key**(*obj*, *callback=None*)

This method is like *pydash.arrays.find_index()* except that it returns the key of the first element that passes the callback check, instead of the element itself.

Parameters

- **obj** (*list|dict*) – Object to search.
- **callback** (*mixed*) – Callback applied per iteration.

Returns Found key or `None`.

Return type mixed

Example

```
>>> find_key({'a': 1, 'b': 2, 'c': 3}, lambda x: x == 1)
'a'
>>> find_key([1, 2, 3, 4], lambda x: x == 1)
0
```

See also:

- *find_key()* (main definition)
- *find_last_key()* (alias)

New in version 1.0.0.

pydash.objects.**find_last_key**(*obj*, *callback=None*)

This method is like *pydash.arrays.find_index()* except that it returns the key of the first element that passes the callback check, instead of the element itself.

Parameters

- **obj** (*list|dict*) – Object to search.
- **callback** (*mixed*) – Callback applied per iteration.

Returns Found key or `None`.

Return type mixed

Example

```
>>> find_key({'a': 1, 'b': 2, 'c': 3}, lambda x: x == 1)
'a'
>>> find_key([1, 2, 3, 4], lambda x: x == 1)
0
```

See also:

- *find_key()* (main definition)
- *find_last_key()* (alias)

New in version 1.0.0.

pydash.objects.**for_in**(*obj*, *callback=None*)

Iterates over own and inherited enumerable properties of *obj*, executing *callback* for each property.

Parameters

- **obj** (*list|dict*) – Object to process.
- **callback** (*mixed*) – Callback applied per iteration.

Returns *obj*.

Return type list|dict

Example

```
>>> obj = {}
>>> def cb(v, k): obj[k] = v
>>> results = for_in({'a': 1, 'b': 2, 'c': 3}, cb)
>>> results == {'a': 1, 'b': 2, 'c': 3}
True
>>> obj == {'a': 1, 'b': 2, 'c': 3}
True
```

See also:

- *for_in()* (main definition)
- *for_own()* (alias)

New in version 1.0.0.

pydash.objects.**for_in_right**(*obj*, *callback=None*)

This function is like *for_in()* except it iterates over the properties in reverse order.

Parameters

- **obj** (*list|dict*) – Object to process.
- **callback** (*mixed*) – Callback applied per iteration.

Returns *obj*.

Return type list|dict

Example

```
>>> data = {'product': 1}
>>> def cb(v): data['product'] *= v
>>> for_in_right([1, 2, 3, 4], cb)
[1, 2, 3, 4]
>>> data['product'] == 24
True
```

See also:

- *for_in_right()* (main definition)
- *for_own_right()* (alias)

New in version 1.0.0.

pydash.objects.**for_own**(*obj*, *callback=None*)

Iterates over own and inherited enumerable properties of *obj*, executing *callback* for each property.

Parameters

- **obj** (*list|dict*) – Object to process.
- **callback** (*mixed*) – Callback applied per iteration.

Returns *obj*.

Return type list|dict

Example

```
>>> obj = {}
>>> def cb(v, k): obj[k] = v
>>> results = for_in({'a': 1, 'b': 2, 'c': 3}, cb)
>>> results == {'a': 1, 'b': 2, 'c': 3}
True
>>> obj == {'a': 1, 'b': 2, 'c': 3}
True
```

See also:

- *for_in()* (main definition)
- *for_own()* (alias)

New in version 1.0.0.

pydash.objects.**for_own_right**(*obj*, *callback=None*)

This function is like *for_in()* except it iterates over the properties in reverse order.

Parameters

- **obj** (*list|dict*) – Object to process.
- **callback** (*mixed*) – Callback applied per iteration.

Returns *obj*.

Return type list|dict

Example

```
>>> data = {'product': 1}
>>> def cb(v): data['product'] *= v
>>> for_in_right([1, 2, 3, 4], cb)
[1, 2, 3, 4]
>>> data['product'] == 24
True
```

See also:

- *for_in_right()* (main definition)
- *for_own_right()* (alias)

New in version 1.0.0.

pydash.objects.**get**(*obj*, *path*, *default=None*)

Get the value at any depth of a nested object based on the path described by *path*. If path doesn't exist, *default* is returned.

Parameters

- **obj** (*list|dict*) – Object to process.
- **path** (*str|list*) – List or `.` delimited string of path describing path.

Keyword Arguments **default** (*mixed*) – Default value to return if path doesn't exist. Defaults to `None`.

Returns Value of *obj* at path.

Return type mixed

Example

```
>>> get({}, 'a.b.c') is None
True
>>> get({'a': {'b': {'c': [1, 2, 3, 4]}}}, 'a.b.c.1')
2
>>> get({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.1')
2
>>> get({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.2') is None
True
```

See also:

- *get()* (main definition)
- *get_path()* (alias)
- *deep_get()* (alias)

New in version 2.0.0.

Changed in version 2.2.0: Support escaping "." delimiter in single string path key.

Changed in version 3.3.0: Added *get()* as main definition and *get_path()* as alias. Made *deep_get()* an alias.

Changed in version 3.4.7: Fixed bug where an iterable default was iterated over instead of being returned when an object path wasn't found.

pydash.objects.**get_path**(*obj*, *path*, *default=None*)

Get the value at any depth of a nested object based on the path described by *path*. If path doesn't exist, *default* is returned.

Parameters

- **obj** (*list|dict*) – Object to process.
- **path** (*str|list*) – List or `.` delimited string of path describing path.

Keyword Arguments **default** (*mixed*) – Default value to return if path doesn't exist. Defaults to `None`.

Returns Value of *obj* at path.

Return type mixed

Example

```
>>> get({}, 'a.b.c') is None
True
>>> get({'a': {'b': {'c': [1, 2, 3, 4]}}}, 'a.b.c.1')
2
>>> get({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.1')
2
>>> get({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.2') is None
True
```

See also:

- *get()* (main definition)
- *get_path()* (alias)
- *deep_get()* (alias)

New in version 2.0.0.

Changed in version 2.2.0: Support escaping "." delimiter in single string path key.

Changed in version 3.3.0: Added *get()* as main definition and *get_path()* as alias. Made *deep_get()* an alias.

Changed in version 3.4.7: Fixed bug where an iterable default was iterated over instead of being returned when an object path wasn't found.

pydash.objects.**has**(*obj*, *path*)

Checks if *path* exists as a key of *obj*.

Parameters

- **obj** (*mixed*) – Object to test.
- **path** (*mixed*) – Path to test for. Can be a list of nested keys or a `.` delimited string of path describing the path.

Returns Whether *obj* has *path*.

Return type bool

Example

```
>>> has([1, 2, 3], 1)
True
>>> has({'a': 1, 'b': 2}, 'b')
True
>>> has({'a': 1, 'b': 2}, 'c')
False
>>> has({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.1')
True
>>> has({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.2')
False
```

See also:

- *has()* (main definition)
- *deep_has()* (alias)
- *has_path()* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Return `False` on `ValueError` when checking path.

Changed in version 3.3.0: Added *deep_has()* as alias. Added *has_path()* as alias.

pydash.objects.**has_path**(*obj*, *path*)

Checks if *path* exists as a key of *obj*.

Parameters

- **obj** (*mixed*) – Object to test.
- **path** (*mixed*) – Path to test for. Can be a list of nested keys or a `.` delimited string of path describing the path.

Returns Whether *obj* has *path*.

Return type bool

Example

```
>>> has([1, 2, 3], 1)
True
>>> has({'a': 1, 'b': 2}, 'b')
True
>>> has({'a': 1, 'b': 2}, 'c')
False
>>> has({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.1')
True
>>> has({'a': {'b': [0, {'c': [1, 2]}]}}, 'a.b.1.c.2')
False
```

See also:

- *has()* (main definition)
- *deep_has()* (alias)
- *has_path()* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Return `False` on `ValueError` when checking path.

Changed in version 3.3.0: Added *deep_has()* as alias. Added *has_path()* as alias.

pydash.objects.**invert**(*obj*, *multivalue=False*)

Creates an object composed of the inverted keys and values of the given object.

Parameters

- **obj** (*dict*) – Dict to invert.
- **multivalue** (*bool, optional*) – Whether to return inverted values as lists. Defaults to `False`.

Returns Inverted dict.

Return type dict

Example

```
>>> results = invert({'a': 1, 'b': 2, 'c': 3})
>>> results == {1: 'a', 2: 'b', 3: 'c'}
True
>>> results = invert({'a': 1, 'b': 2, 'c': 1}, multivalue=True)
>>> set(results[1]) == set(['a', 'c'])
True
```

Note: Assumes *dict* values are hashable as *dict* keys.

New in version 1.0.0.

Changed in version 2.0.0: Added `multivalue` argument.

`pydash.objects.`**`keys`**(*obj*)

Creates a list composed of the keys of *obj*.

Parameters **`obj`** (*mixed*) – Object to extract keys from.

Returns List of keys.

Return type list

Example

```
>>> keys([1, 2, 3])
[0, 1, 2]
>>> set(keys({'a': 1, 'b': 2, 'c': 3})) == set(['a', 'b', 'c'])
True
```

See also:

- *keys()* (main definition)
- *keys_in()* (alias)

New in version 1.0.0.

Changed in version 1.1.0: Added *keys_in()* as alias.

`pydash.objects.`**`keys_in`**(*obj*)

Creates a list composed of the keys of *obj*.

Parameters **`obj`** (*mixed*) – Object to extract keys from.

Returns List of keys.

Return type list

Example

```
>>> keys([1, 2, 3])
[0, 1, 2]
>>> set(keys({'a': 1, 'b': 2, 'c': 3})) == set(['a', 'b', 'c'])
True
```

See also:

- *keys()* (main definition)
- *keys_in()* (alias)

New in version 1.0.0.

Changed in version 1.1.0: Added *keys_in()* as alias.

pydash.objects.**map_keys**(*obj*, *callback=None*)

Creates an object with the same values as *obj* and keys generated by running each property of *obj* through the *callback*. The callback is invoked with three arguments: `(value, key, object)`. If a property name is provided for *callback* the created *pydash.collections.pluck()* style callback will return the property value of the given element. If an object is provided for callback the created *pydash.collections.where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **obj** (*list|dict*) – Object to map.
- **callback** (*mixed*) – Callback applied per iteration.

Returns Results of running *obj* through *callback*.

Return type list|dict

Example

```
>>> callback = lambda value, key: key * 2
>>> results = map_keys({'a': 1, 'b': 2, 'c': 3}, callback)
>>> results == {'aa': 1, 'bb': 2, 'cc': 3}
True
```

New in version 3.3.0.

pydash.objects.**map_values**(*obj*, *callback=None*)

Creates an object with the same keys as *obj* and values generated by running each property of *obj* through the *callback*. The callback is invoked with three arguments: `(value, key, object)`. If a property name is provided for *callback* the created *pydash.collections.pluck()* style callback will return the property value of the given element. If an object is provided for callback the created *pydash.collections.where()* style callback will return `True` for elements that have the properties of the given object, else `False`.

Parameters

- **obj** (*list|dict*) – Object to map.
- **callback** (*mixed*) – Callback applied per iteration.

Returns Results of running *obj* through *callback*.

Return type list|dict

Example

```
>>> results = map_values({'a': 1, 'b': 2, 'c': 3}, lambda x: x * 2)
>>> results == {'a': 2, 'b': 4, 'c': 6}
True
>>> results = map_values({'a': 1, 'b': {'d': 4}, 'c': 3}, {'d': 4})
>>> results == {'a': False, 'b': True, 'c': False}
True
```

New in version 1.0.0.

pydash.objects.**merge**(*obj*, **sources*, ***kargs*)

Recursively merges own enumerable properties of the source object(s) that don't resolve to undefined into the destination object. Subsequent sources will overwrite property assignments of previous sources. If a callback is

provided it will be executed to produce the merged values of the destination and source properties. The callback is invoked with at least two arguments: `(obj_value, *source_value)`.

Parameters

- **obj** (*dict*) – Destination object to merge source(s) into.
- **sources** (*dict*) – Source objects to merge from. subsequent sources overwrite previous ones.

Keyword Arguments **callback** (*function, optional*) – Callback function to handle merging (must be passed in as keyword argument).

Returns Merged object.

Return type dict

Warning: *obj* is modified in place.

Example

```
>>> obj = {'a': 2}
>>> obj2 = merge(obj, {'a': 1}, {'b': 2, 'c': 3}, {'d': 4})
>>> obj2 == {'a': 1, 'b': 2, 'c': 3, 'd': 4}
True
>>> obj is obj2
True
```

New in version 1.0.0.

Changed in version 2.3.2: Apply *clone_deep()* to each *source* before assigning to *obj*.

Changed in version 2.3.2: Allow *callback* to be passed by reference if it is the last positional argument.

Changed in version 3.3.0: Added internal option for overriding the default setter for obj values.

pydash.objects.**methods**(*obj*)

Creates a sorted list of keys of an object that are callable.

Parameters **obj** (*list|dict*) – Object to inspect.

Returns All keys whose values are callable.

Return type list

Example

```
>>> callables({'a': 1, 'b': lambda: 2, 'c': lambda: 3})
['b', 'c']
```

See also:

- *callables()* (main definition)
- *methods()* (alias)

New in version 1.0.0.

Changed in version 2.0.0: Renamed `functions` to `callables`.

pydash.objects.**omit**(*obj*, *callback=None*, **properties*)

Creates a shallow clone of object excluding the specified properties. Property names may be specified as individual arguments or as lists of property names. If a callback is provided it will be executed for each property of object omitting the properties the callback returns truthy for. The callback is invoked with three arguments: `(value, key, object)`.

Parameters

- **obj** (*mixed*) – Object to process.
- **properties** (*str*) – Property values to omit.
- **callback** (*mixed, optional*) – Callback used to determine whic properties to omit.

Returns Results of omitting properties.

Return type dict

Example

```
>>> omit({'a': 1, 'b': 2, 'c': 3}, 'b', 'c') == {'a': 1}
True
>>> omit([1, 2, 3, 4], 0, 3) == {1: 2, 2: 3}
True
```

New in version 1.0.0.

pydash.objects.**pairs**(*obj*)

Creates a two dimensional list of an object's key-value pairs, i.e. `[[key1, value1], [key2, value2]]`.

Parameters **obj** (*mixed*) – Object to process.

Returns Two dimensional list of object's key-value pairs.

Return type list

Example

```
>>> pairs([1, 2, 3, 4])
[[0, 1], [1, 2], [2, 3], [3, 4]]
>>> pairs({'a': 1})
[['a', 1]]
```

New in version 1.0.0.

pydash.objects.**parse_int**(*value*, *radix=None*)

Converts the given *value* into an integer of the specified *radix*. If *radix* is falsey, a radix of `10` is used unless the *value* is a hexadecimal, in which case a radix of 16 is used.

Parameters

- **value** (*mixed*) – Value to parse.
- **radix** (*int, optional*) – Base to convert to.

Returns Integer if parsable else `None`.

Return type mixed

Example

```
>>> parse_int('5')
5
>>> parse_int('12', 8)
10
>>> parse_int('x') is None
True
```

New in version 1.0.0.

pydash.objects.**pick**(*obj*, *callback=None*, **properties*)

Creates a shallow clone of object composed of the specified properties. Property names may be specified as individual arguments or as lists of property names. If a callback is provided it will be executed for each property of object picking the properties the callback returns truthy for. The callback is invoked with three arguments: `(value, key, object)`.

Parameters

- **obj** (*list|dict*) – Object to pick from.
- **properties** (*str*) – Property values to pick.
- **callback** (*mixed, optional*) – Callback used to determine which properties to pick.

Returns Dict containg picked properties.

Return type dict

Example

```
>>> pick({'a': 1, 'b': 2, 'c': 3}, 'a', 'b') == {'a': 1, 'b': 2}
True
```

New in version 1.0.0.

pydash.objects.**rename_keys**(*obj*, *key_map*)

Rename the keys of *obj* using *key_map* and return new object.

Parameters

- **obj** (*dict*) – Object to rename.
- **key_map** (*dict*) – Renaming map whose keys correspond to existing keys in *obj* and whose values are the new key name.

Returns Renamed *obj*.

Return type dict

Example

```
>>> obj = rename_keys({'a': 1, 'b': 2, 'c': 3}, {'a': 'A', 'b': 'B'})
>>> obj == {'A': 1, 'B': 2, 'c': 3}
True
```

New in version 2.0.0.

pydash.objects.**set_**(*obj*, *path*, *value*)

Sets the value of an object described by *path*. If any part of the object path doesn't exist, it will be created.

Parameters

- **obj** (*list|dict*) – Object to modify.
- **path** (*str | list*) – Target path to set value to.
- **value** (*mixed*) – Value to set.

Returns Modified *obj*.

Return type mixed

Example

```
>>> set_({}, 'a.b.c', 1)
{'a': {'b': {'c': 1}}}
>>> set_({}, 'a.0.c', 1)
{'a': {'0': {'c': 1}}}
>>> set_([1, 2], '2.0', 1)
[1, 2, [1]]
```

New in version 2.2.0.

Changed in version 3.3.0: Added *set_()* as main definition and *deep_set()* as alias.

pydash.objects.**set_path**(*obj*, *value*, *keys*, *default=None*)

Sets the value of an object described by *keys*. If any part of the object path doesn't exist, it will be created with *default*.

Parameters

- **obj** (*list|dict*) – Object to modify.
- **value** (*mixed*) – Value to set.
- **keys** (*list*) – Target path to set value to.
- **default** (*callable, optional*) – Callable that returns default value to assign if path part is not set. Defaults to `{}` if *obj* is a `dict` or `[]` if *obj* is a `list`.

Returns Modified *obj*.

Return type mixed

Example

```
>>> set_path({}, 1, ['a', 0], default=[])
{'a': [1]}
>>> set_path({}, 1, ['a', 'b']) == {'a': {'b': 1}}
True
```

New in version 2.0.0.

pydash.objects.**to_boolean**(*obj*, *true_values=('true', '1')*, *false_values=('false', '0')*)

Convert *obj* to boolean. This is not like the builtin `bool` function. By default commonly considered strings values are converted to their boolean equivalent, i.e., `'0'` and `'false'` are converted to `False` while `'1'` and `'true'` are converted to `True`. If a string value is provided that isn't recognized as having a common boolean conversion, then the returned value is `None`. Non-string values of *obj* are converted using `bool`. Optionally, *true_values* and *false_values* can be overridden but each value must be a string.

Parameters

- **obj** (*mixed*) – Object to convert.
- **true_values** (*tuple, optional*) – Values to consider `True`. Each value must be a string. Comparision is case-insensitive. Defaults to `('true', '1')`.
- **false_values** (*tuple, optional*) – Values to consider `False`. Each value must be a string. Comparision is case-insensitive. Defaults to `('false', '0')`.

Returns Boolean value of *obj*.

Return type bool

Example

```
>>> to_boolean('true')
True
>>> to_boolean('1')
True
>>> to_boolean('false')
False
>>> to_boolean('0')
False
>>> assert to_boolean('a') is None
```

New in version 3.0.0.

pydash.objects.**to_dict**(*obj*)

Convert *obj* to `dict` by created a new `dict` using *obj* keys and values.

Parameters **obj** – (mixed): Object to convert.

Returns Object converted to `dict`.

Return type dict

Example

```
>>> obj = {'a': 1, 'b': 2}
>>> obj2 = to_dict(obj)
>>> obj2 == obj
True
>>> obj2 is not obj
True
```

New in version 3.0.0.

pydash.objects.**to_number**(*obj*, *precision=0*)

Convert *obj* to a number. All numbers are retuned as `float`. If precision is negative, round *obj* to the nearest positive integer place. If *obj* can't be converted to a number, `None` is returned.

Parameters

- **obj** (*str|int|float*) – Object to convert.
- **precision** (*int, optional*) – Precision to round number to. Defaults to `0`.

Returns Converted number or `None` if can't be converted.

Return type float

Example

```
>>> to_number('1234.5678')
1235.0
>>> to_number('1234.5678', 4)
1234.5678
>>> to_number(1, 2)
1.0
```

New in version 3.0.0.

pydash.objects.**to_plain_object**(*obj*)

Convert *obj* to `dict` by created a new `dict` using *obj* keys and values.

Parameters **obj** – (mixed): Object to convert.

Returns Object converted to `dict`.

Return type dict

Example

```
>>> obj = {'a': 1, 'b': 2}
>>> obj2 = to_dict(obj)
>>> obj2 == obj
True
>>> obj2 is not obj
True
```

New in version 3.0.0.

pydash.objects.**to_string**(*obj*)

Converts an object to string.

Parameters **obj** (*mixed*) – Object to convert.

Returns String representation of *obj*.

Return type str

Example

```
>>> to_string(1) == '1'
True
>>> to_string(None) == ''
True
>>> to_string([1, 2, 3]) == '[1, 2, 3]'
True
>>> to_string('a') == 'a'
True
```

New in version 2.0.0.

Changed in version 3.0.0: Convert `None` to empty string.

pydash.objects.**transform**(*obj*, *callback=None*, *accumulator=None*)

An alernative to `pydash.collections.reduce()`, this method transforms *obj* to a new accumulator object which is the result of running each of its properties through a callback, with each callback execution

potentially mutating the accumulator object. The callback is invoked with four arguments: `(accumulator, value, key, object)`. Callbacks may exit iteration early by explicitly returning `False`.

Parameters

- **obj** (*list|dict*) – Object to process.
- **callback** (*mixed*) – Callback applied per iteration.
- **accumulator** (*mixed, optional*) – Accumulated object. Defaults to `list`.

Returns Accumulated object.

Return type mixed

Example

```
>>> transform([1, 2, 3, 4], lambda acc, value, key: acc.append((key, value)
[(0, 1), (1, 2), (2, 3), (3, 4)]
```

New in version 1.0.0.

pydash.objects.**update_path**(*obj*, *callback*, *keys*, *default=None*)

Update the value of an object described by *keys* using *callback*. If any part of the object path doesn't exist, it will be created with *default*. The callback is invoked with the last key value of *obj*: `(value)`

Parameters

- **obj** (*list|dict*) – Object to modify.
- **callback** (*function*) – Function that returns updated value.
- **keys** (*list*) – A list of string keys that describe the object path to modify.
- **default** (*mixed, optional*) – Default value to assign if path part is not set. Defaults to `{}` if *obj* is a `dict` or `[]` if *obj* is a `list`.

Returns Updated *obj*.

Return type mixed

Example

```
>>> update_path({}, lambda value: value, ['a', 'b'])
{'a': {'b': None}}
>>> update_path([], lambda value: value, [0, 0])
[[None]]
```

New in version 2.0.0.

pydash.objects.**values**(*obj*)

Creates a list composed of the values of *obj*.

Parameters **obj** (*mixed*) – Object to extract values from.

Returns List of values.

Return type list

Example

```
>>> results = values({'a': 1, 'b': 2, 'c': 3})
>>> set(results) == set([1, 2, 3])
True
>>> values([2, 4, 6, 8])
[2, 4, 6, 8]
```

See also:

- *values()* (main definition)
- *values_in()* (alias)

New in version 1.0.0.

Changed in version 1.1.0: Added *values_in()* as alias.

pydash.objects.**values_in**(*obj*)

Creates a list composed of the values of *obj*.

Parameters **obj** (*mixed*) – Object to extract values from.

Returns List of values.

Return type list

Example

```
>>> results = values({'a': 1, 'b': 2, 'c': 3})
>>> set(results) == set([1, 2, 3])
True
>>> values([2, 4, 6, 8])
[2, 4, 6, 8]
```

See also:

- *values()* (main definition)
- *values_in()* (alias)

New in version 1.0.0.

Changed in version 1.1.0: Added *values_in()* as alias.

4.1.8 Predicates

Predicate functions that return boolean evaluations of objects.

New in version 2.0.0.

pydash.predicates.**gt**(*value*, *other*)

Checks if *value* is greater than *other*.

Parameters

- **value** (*number*) – Value to compare.
- **other** (*number*) – Other value to compare.

Returns Whether *value* is greater than *other*.

Return type bool

Example

```
>>> gt(5, 3)
True
>>> gt(3, 5)
False
>>> gt(5, 5)
False
```

New in version 3.3.0.

pydash.predicates.**gte**(*value*, *other*)
Checks if *value* is greater than or equal to *other*.

Parameters

- **value** (*number*) – Value to compare.
- **other** (*number*) – Other value to compare.

Returns Whether *value* is greater than or equal to *other*.

Return type bool

Example

```
>>> gte(5, 3)
True
>>> gte(3, 5)
False
>>> gte(5, 5)
True
```

New in version 3.3.0.

pydash.predicates.**lt**(*value*, *other*)
Checks if *value* is less than *other*.

Parameters

- **value** (*number*) – Value to compare.
- **other** (*number*) – Other value to compare.

Returns Whether *value* is less than *other*.

Return type bool

Example

```
>>> lt(5, 3)
False
>>> lt(3, 5)
True
```

```
    >>> lt(5, 5)
    False
```

New in version 3.3.0.

pydash.predicates.**lte**(*value*, *other*)

Checks if *value* is less than or equal to *other*.

Parameters

- **value** (*number*) – Value to compare.
- **other** (*number*) – Other value to compare.

Returns Whether *value* is less than or equal to *other*.

Return type bool

Example

```
    >>> lte(5, 3)
    False
    >>> lte(3, 5)
    True
    >>> lte(5, 5)
    True
```

New in version 3.3.0.

pydash.predicates.**in_range**(*value*, *start=0*, *end=None*)

Checks if *value* is between *start* and up to but not including *end*. If *end* is not specified it defaults to *start* with *start* becoming `0`.

Parameters

- **value** (*int|float*) – Number to check.
- **start** (*int|float, optional*) – Start of range inclusive. Defaults to `0`.
- **end** (*int|float, optional*) – End of range exclusive. Defaults to *start*.

Returns Whether *value* is in range.

Return type bool

Example

```
    >>> in_range(2, 4)
    True
    >>> in_range(4, 2)
    False
    >>> in_range(2, 1, 3)
    True
    >>> in_range(3, 1, 2)
    False
    >>> in_range(2.5, 3.5)
    True
    >>> in_range(3.5, 2.5)
    False
```

New in version 3.1.0.

pydash.predicates.**is_associative**(*value*)

Checks if *value* is an associative object meaning that it can be accessed via an index or key

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is associative.

Return type bool

Example

```
>>> is_associative([])
True
>>> is_associative({})
True
>>> is_associative(1)
False
>>> is_associative(True)
False
```

New in version 2.0.0.

pydash.predicates.**is_blank**(*text*)

Checks if *text* contains only whitespace characters.

Parameters **text** (*str*) – String to test.

Returns Whether *text* is blank.

Return type bool

Example

```
>>> is_blank('')
True
>>> is_blank(' \r\n ')
True
>>> is_blank(False)
False
```

New in version 3.0.0.

pydash.predicates.**is_boolean**(*value*)

Checks if *value* is a boolean value.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a boolean.

Return type bool

Example

```
>>> is_boolean(True)
True
>>> is_boolean(False)
True
```

```
>>> is_boolean(0)
False
```

See also:

- *is_boolean()* (main definition)
- *is_bool()* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Added `is_bool` as alias.

`pydash.predicates.`**`is_bool`**`(`*value*`)`

Checks if *value* is a boolean value.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a boolean.

Return type bool

Example

```
>>> is_boolean(True)
True
>>> is_boolean(False)
True
>>> is_boolean(0)
False
```

See also:

- *is_boolean()* (main definition)
- *is_bool()* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Added `is_bool` as alias.

`pydash.predicates.`**`is_builtin`**`(`*value*`)`

Checks if *value* is a Python builtin function or method.

Parameters **value** (*function*) – Value to check.

Returns Whether *value* is a Python builtin function or method.

Return type bool

Example

```
>>> is_builtin(1)
True
>>> is_builtin(list)
True
>>> is_builtin('foo')
False
```

See also:

- *is_builtin()* (main definition)
- *is_native()* (alias)

New in version 3.0.0.

pydash.predicates.**is_date**(*value*)
Check if *value* is a date object.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a date object.

Return type bool

Example

```
>>> import datetime
>>> is_date(datetime.date.today())
True
>>> is_date(datetime.datetime.today())
True
>>> is_date('2014-01-01')
False
```

Note: This will also return `True` for datetime objects.

New in version 1.0.0.

pydash.predicates.**is_decreasing**(*value*)
Check if *value* is monotonically decreasing.

Parameters **value** (*list*) – Value to check.

Returns Whether *value* is monotonically decreasing.

Return type bool

Example

```
>>> is_decreasing([5, 4, 4, 3])
True
>>> is_decreasing([5, 5, 5])
True
>>> is_decreasing([5, 4, 5])
False
```

New in version 2.0.0.

pydash.predicates.**is_dict**(*value*)
Checks if *value* is a `dict`.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a `dict`.

Return type bool

Example

```
>>> is_dict({})
True
>>> is_dict([])
False
```

See also:

- *is_dict()* (main definition)
- *is_plain_object()* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Added *is_dict()* as main definition and made *is_plain_object()* an alias.

pydash.predicates.**is_empty**(*value*)

Checks if *value* is empty.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is empty.

Return type bool

Example

```
>>> is_empty(0)
True
>>> is_empty(1)
True
>>> is_empty(True)
True
>>> is_empty('foo')
False
>>> is_empty(None)
True
>>> is_empty({})
True
```

Note: Returns `True` for booleans and numbers.

New in version 1.0.0.

pydash.predicates.**is_equal**(*value*, *other*, *callback=None*)

Performs a comparison between two values to determine if they are equivalent to each other. If a callback is provided it will be executed to compare values. If the callback returns `None`, comparisons will be handled by the method instead. The callback is invoked with two arguments: `(value, other)`.

Parameters

- **value** (*list|dict*) – Object to compare.
- **other** (*list|dict*) – Object to compare.

- **callback** (*mixed, optional*) – Callback used to compare values from *value* and *other*.

Returns Whether *value* and *other* are equal.

Return type bool

Example

```
>>> is_equal([1, 2, 3], [1, 2, 3])
True
>>> is_equal('a', 'A')
False
>>> is_equal('a', 'A', lambda a, b: a.lower() == b.lower())
True
```

New in version 1.0.0.

pydash.predicates.**is_error**(*value*)

Checks if *value* is an `Exception`.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is an exception.

Return type bool

Example

```
>>> is_error(Exception())
True
>>> is_error(Exception)
False
>>> is_error(None)
False
```

New in version 1.1.0.

pydash.predicates.**is_even**(*value*)

Checks if *value* is even.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is even.

Return type bool

Example

```
>>> is_even(2)
True
>>> is_even(3)
False
>>> is_even(False)
False
```

New in version 2.0.0.

pydash.predicates.**is_float**(*value*)
Checks if *value* is a float.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a float.

Return type bool

Example

```
>>> is_float(1.0)
True
>>> is_float(1)
False
```

New in version 2.0.0.

pydash.predicates.**is_function**(*value*)
Checks if *value* is a function.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is callable.

Return type bool

Example

```
>>> is_function(list)
True
>>> is_function(lambda: True)
True
>>> is_function(1)
False
```

New in version 1.0.0.

pydash.predicates.**is_increasing**(*value*)
Check if *value* is monotonically increasing.

Parameters **value** (*list*) – Value to check.

Returns Whether *value* is monotonically increasing.

Return type bool

Example

```
>>> is_increasing([1, 3, 5])
True
>>> is_increasing([1, 1, 2, 3, 3])
True
>>> is_increasing([5, 5, 5])
True
>>> is_increasing([1, 2, 4, 3])
False
```

New in version 2.0.0.

pydash.predicates.**is_indexed**(*value*)
Checks if *value* is integer indexed, i.e., `list`, `str` or `tuple`.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is integer indexed.

Return type bool

Example

```
>>> is_indexed('')
True
>>> is_indexed([])
True
>>> is_indexed(())
True
>>> is_indexed({})
False
```

New in version 2.0.0.

Changed in version 3.0.0: Return `True` for tuples.

pydash.predicates.**is_instance_of**(*value*, *types*)
Checks if *value* is an instance of *types*.

Parameters

- **value** (*mixed*) – Value to check.
- **types** (*mixed*) – Types to check against. Pass as `tuple` to check if *value* is one of multiple types.

Returns Whether *value* is an instance of *types*.

Return type bool

Example

```
>>> is_instance_of({}, dict)
True
>>> is_instance_of({}, list)
False
```

New in version 2.0.0.

pydash.predicates.**is_integer**(*value*)
Checks if *value* is a integer.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is an integer.

Return type bool

Example

```
>>> is_integer(1)
True
>>> is_integer(1.0)
False
>>> is_integer(True)
False
```

See also:

- *is_integer()* (main definition)
- *is_int()* (alias)

New in version 2.0.0.

Changed in version 3.0.0: Added `is_int` as alias.

`pydash.predicates.`**`is_int`**`(`*value*`)`

Checks if *value* is a integer.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is an integer.

Return type bool

Example

```
>>> is_integer(1)
True
>>> is_integer(1.0)
False
>>> is_integer(True)
False
```

See also:

- *is_integer()* (main definition)
- *is_int()* (alias)

New in version 2.0.0.

Changed in version 3.0.0: Added `is_int` as alias.

`pydash.predicates.`**`is_iterable`**`(`*value*`)`

Checks if *value* is an iterable.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is an iterable.

Return type bool

Example

```
>>> is_iterable([])
True
>>> is_iterable({})
True
>>> is_iterable(())
True
>>> is_iterable(5)
False
>>> is_iterable(True)
False
```

New in version 3.3.0.

pydash.predicates.**is_json**(*value*)
Checks if *value* is a valid JSON string.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is JSON.

Return type bool

Example

```
>>> is_json({})
False
>>> is_json('{}')
True
>>> is_json({"hello": 1, "world": 2})
False
>>> is_json('{"hello": 1, "world": 2}')
True
```

New in version 2.0.0.

pydash.predicates.**is_list**(*value*)
Checks if *value* is a list.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a list.

Return type bool

Example

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

New in version 1.0.0.

pydash.predicates.**is_match**(*obj*, *source*, *callback=None*)
Performs a comparison between *obj* and *source* to determine if *obj* contains equivalent property values as *source*.

If a callback is provided it will be executed to compare values. If the callback returns `None`, comparisons will be handled by the method instead. The callback is invoked with two arguments: `(obj, source)`.

Parameters

- **obj** (*list|dict*) – Object to compare.
- **source** (*list|dict*) – Object of property values to match.
- **callback** (*mixed, optional*) – Callback used to compare values from *obj* and *source*.

Returns Whether *obj* is a match or not.

Return type bool

Example

```
>>> is_match({'a': 1, 'b': 2}, {'b': 2})
True
>>> is_match({'a': 1, 'b': 2}, {'b': 3})
False
>>> is_match({'a': [{'b': [{'c': 3, 'd': 4}]}]}, {'a': [{'b': [{'d': 4}]}]})
True
```

New in version 3.0.0.

Changed in version 3.2.0: Don't compare *obj* and *source* using `type`. Use `isinstance` exclusively.

`pydash.predicates.`**is_monotone**(*value*, *op*)

Checks if *value* is monotonic when *operator* used for comparison.

Parameters

- **value** (*list*) – Value to check.
- **op** (*function*) – Operation to used for comparison.

Returns Whether *value* is monotone.

Return type bool

Example

```
>>> is_monotone([1, 1, 2, 3], operator.le)
True
>>> is_monotone([1, 1, 2, 3], operator.lt)
False
```

New in version 2.0.0.

`pydash.predicates.`**is_nan**(*value*)

Checks if *value* is not a number.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is not a number.

Return type bool

Example

```
>>> is_nan('a')
True
>>> is_nan(1)
False
>>> is_nan(1.0)
False
```

New in version 1.0.0.

pydash.predicates.**is_native**(*value*)

Checks if *value* is a Python builtin function or method.

Parameters **value** (*function*) – Value to check.

Returns Whether *value* is a Python builtin function or method.

Return type bool

Example

```
>>> is_builtin(1)
True
>>> is_builtin(list)
True
>>> is_builtin('foo')
False
```

See also:

- *is_builtin()* (main definition)
- *is_native()* (alias)

New in version 3.0.0.

pydash.predicates.**is_negative**(*value*)

Checks if *value* is negative.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is negative.

Return type bool

Example

```
>>> is_negative(-1)
True
>>> is_negative(0)
False
>>> is_negative(1)
False
```

New in version 2.0.0.

pydash.predicates.**is_none**(*value*)

Checks if *value* is *None*.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is `None`.

Return type bool

Example

```
>>> is_none(None)
True
>>> is_none(False)
False
```

New in version 1.0.0.

`pydash.predicates.`**`is_number`**`(`*value*`)`

Checks if *value* is a number.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a number.

Return type bool

Note: Returns `True` for `int`, `long` (PY2), `float`, and `decimal.Decimal`.

Example

```
>>> is_number(1)
True
>>> is_number(1.0)
True
>>> is_number('a')
False
```

See also:

- *`is_number()`* (main definition)
- *`is_num()`* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Added `is_num` as alias.

`pydash.predicates.`**`is_num`**`(`*value*`)`

Checks if *value* is a number.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a number.

Return type bool

Note: Returns `True` for `int`, `long` (PY2), `float`, and `decimal.Decimal`.

Example

```
>>> is_number(1)
True
>>> is_number(1.0)
True
>>> is_number('a')
False
```

See also:

- *is_number()* (main definition)
- *is_num()* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Added `is_num` as alias.

`pydash.predicates.`**`is_object`**`(`*value*`)`

Checks if *value* is a `list` or `dict`.

Parameters **`value`** (*mixed*) – Value to check.

Returns Whether *value* is `list` or `dict`.

Return type bool

Example

```
>>> is_object([])
True
>>> is_object({})
True
>>> is_object(())
False
>>> is_object(1)
False
```

New in version 1.0.0.

`pydash.predicates.`**`is_odd`**`(`*value*`)`

Checks if *value* is odd.

Parameters **`value`** (*mixed*) – Value to check.

Returns Whether *value* is odd.

Return type bool

Example

```
>>> is_odd(3)
True
>>> is_odd(2)
False
>>> is_odd('a')
False
```

New in version 2.0.0.

pydash.predicates.**is_plain_object**(*value*)
Checks if *value* is a `dict`.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a `dict`.

Return type bool

Example

```
>>> is_dict({})
True
>>> is_dict([])
False
```

See also:

- *is_dict()* (main definition)
- *is_plain_object()* (alias)

New in version 1.0.0.

Changed in version 3.0.0: Added *is_dict()* as main definition and made *is_plain_object()* an alias.

pydash.predicates.**is_positive**(*value*)
Checks if *value* is positive.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is positive.

Return type bool

Example

```
>>> is_positive(1)
True
>>> is_positive(0)
False
>>> is_positive(-1)
False
```

New in version 2.0.0.

pydash.predicates.**is_re**(*value*)
Checks if *value* is a `RegExp` object.

Parameters **value** (*mxied*) – Value to check.

Returns Whether *value* is a RegExp object.

Return type bool

Example

```
>>> is_reg_exp(re.compile(''))
True
>>> is_reg_exp('')
False
```

See also:

- *is_reg_exp()* (main definition)
- *is_re()* (alias)

New in version 1.1.0.

pydash.predicates.**is_reg_exp**(*value*)

Checks if *value* is a `RegExp` object.

Parameters **value** (*mxied*) – Value to check.

Returns Whether *value* is a RegExp object.

Return type bool

Example

```
>>> is_reg_exp(re.compile(''))
True
>>> is_reg_exp('')
False
```

See also:

- *is_reg_exp()* (main definition)
- *is_re()* (alias)

New in version 1.1.0.

pydash.predicates.**is_strictly_decreasing**(*value*)

Check if *value* is strictly decreasing.

Parameters **value** (*list*) – Value to check.

Returns Whether *value* is strictly decreasing.

Return type bool

Example

```
>>> is_strictly_decreasing([4, 3, 2, 1])
True
>>> is_strictly_decreasing([4, 4, 2, 1])
False
```

New in version 2.0.0.

pydash.predicates.**is_strictly_increasing**(*value*)

Check if *value* is strictly increasing.

Parameters **value** (*list*) – Value to check.

Returns Whether *value* is strictly increasing.

Return type bool

Example

```
>>> is_strictly_increasing([1, 2, 3, 4])
True
>>> is_strictly_increasing([1, 1, 3, 4])
False
```

New in version 2.0.0.

pydash.predicates.**is_string**(*value*)
Checks if *value* is a string.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a string.

Return type bool

Example

```
>>> is_string('')
True
>>> is_string(1)
False
```

New in version 1.0.0.

pydash.predicates.**is_tuple**(*value*)
Checks if *value* is a tuple.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is a tuple.

Return type bool

Example

```
>>> is_tuple(())
True
>>> is_tuple({})
False
>>> is_tuple([])
False
```

New in version 3.0.0.

pydash.predicates.**is_zero**(*value*)
Checks if *value* is `0`.

Parameters **value** (*mixed*) – Value to check.

Returns Whether *value* is `0`.

Return type bool

Example

```
>>> is_zero(0)
True
>>> is_zero(1)
False
```

New in version 2.0.0.

4.1.9 Strings

String functions.

New in version 1.1.0.

pydash.strings.**camel_case**(*text*)

Converts *text* to camel case.

Parameters **text** (*str*) – String to convert.

Returns String converted to camel case.

Return type str

Example

```
>>> camel_case('FOO BAR_bAz')
'fooBarBAz'
```

New in version 1.1.0.

pydash.strings.**capitalize**(*text*, *strict=True*)

Capitalizes the first character of *text*.

Parameters

- **text** (*str*) – String to capitalize.
- **strict** (*bool, optional*) – Whether to cast rest of string to lower case. Defaults to `True`.

Returns Capitalized string.

Return type str

Example

```
>>> capitalize('once upon a TIME')
'Once upon a time'
>>> capitalize('once upon a TIME', False)
'Once upon a TIME'
```

New in version 1.1.0.

Changed in version 3.0.0: Added *strict* option.

pydash.strings.**chop**(*text*, *step*)
Break up *text* into intervals of length *step*.

Parameters

- **text** (*str*) – String to chop.
- **step** (*int*) – Interval to chop *text*.

Returns List of chopped characters. If *text* is *None* an empty list is returned.

Return type list

Example

```
>>> chop('abcdefg', 3)
['abc', 'def', 'g']
```

New in version 3.0.0.

pydash.strings.**chop_right**(*text*, *step*)
Like *chop()* except *text* is chopped from right.

Parameters

- **text** (*str*) – String to chop.
- **step** (*int*) – Interval to chop *text*.

Returns List of chopped characters.

Return type list

Example

```
>>> chop_right('abcdefg', 3)
['a', 'bcd', 'efg']
```

New in version 3.0.0.

pydash.strings.**chars**(*text*)
Split *text* into a list of single characters.

Parameters **text** (*str*) – String to split up.

Returns List of individual characters.

Return type list

Example

```
>>> chars('onetwo')
['o', 'n', 'e', 't', 'w', 'o']
```

New in version 3.0.0.

pydash.strings.**clean**(*text*)
Trim and replace multiple spaces with a single space.

Parameters **text** (*str*) – String to clean.

Returns Cleaned string.

Return type str

Example

```
>>> clean('a  b   c    d')
'a b c d'
```

New in version 3.0.0.

pydash.strings.**count_substr**(*text*, *subtext*)
Count the occurrences of *subtext* in *text*.

Parameters

- **text** (*str*) – Source string to count from.
- **subtext** (*str*) – String to count.

Returns Number of occurrences of *subtext* in *text*.

Return type int

Example

```
>>> count_substr('aabbccddaabbccdd', 'bc')
2
```

New in version 3.0.0.

pydash.strings.**deburr**(*text*)
Deburrs *text* by converting latin-1 supplementary letters to basic latin letters.

Parameters **text** (*str*) – String to deburr.

Returns Deburred string.

Return type str

Example

```
>>> deburr('déjà vu')
'...
>>> 'deja vu'
'deja vu'
```

New in version 2.0.0.

pydash.strings.**decapitalize**(*text*)
Decaptitalizes the first character of *text*.

Parameters **text** (*str*) – String to decapitalize.

Returns Decapitalized string.

Return type str

Example

```
>>> decapitalize('FOO BAR')
'fOO BAR'
```

New in version 3.0.0.

pydash.strings.**ends_with**(*text*, *target*, *position=None*)

Checks if *text* ends with a given target string.

Parameters

- **text** (*str*) – String to check.
- **target** (*str*) – String to check for.
- **position** (*int, optional*) – Position to search from. Defaults to end of *text*.

Returns Whether *text* ends with *target*.

Return type bool

Example

```
>>> ends_with('abc def', 'def')
True
>>> ends_with('abc def', 4)
False
```

New in version 1.1.0.

pydash.strings.**ensure_ends_with**(*text*, *suffix*)

Append a given suffix to a string, but only if the source string does not end with that suffix.

Parameters

- **text** (*str*) – Source string to append *suffix* to.
- **suffix** (*str*) – String to append to the source string if the source string does not end with *suffix*.

Returns source string possibly extended by *suffix*.

Return type str

Example

```
>>> ensure_ends_with('foo bar', '!')
'foo bar!'
>>> ensure_ends_with('foo bar!', '!')
'foo bar!'
```

New in version 2.4.0.

pydash.strings.**ensure_starts_with**(*text*, *prefix*)

Prepend a given prefix to a string, but only if the source string does not start with that prefix.

Parameters

- **text** (*str*) – Source string to prepend *prefix* to.

- **suffix** (*str*) – String to prepend to the source string if the source string does not start with *prefix*.

Returns source string possibly prefixed by *prefix*

Return type str

Example

```
>>> ensure_starts_with('foo bar', 'Oh my! ')
'Oh my! foo bar'
>>> ensure_starts_with('Oh my! foo bar', 'Oh my! ')
'Oh my! foo bar'
```

New in version 2.4.0.

pydash.strings.**escape**(*text*)

Converts the characters `&`, `<`, `>`, `"`, `'`, and `` ` `` in *text* to their corresponding HTML entities.

Parameters **text** (*str*) – String to escape.

Returns HTML escaped string.

Return type str

Example

```
>>> escape('"1 > 2 && 3 < 4"')
'&quot;1 &gt; 2 &amp;&amp; 3 &lt; 4&quot;'
```

New in version 1.0.0.

Changed in version 1.1.0: Moved function to *pydash.strings*.

pydash.strings.**escape_reg_exp**(*text*)

Escapes the RegExp special characters in *text*.

Parameters **text** (*str*) – String to escape.

Returns RegExp escaped string.

Return type str

Example

```
>>> escape_reg_exp('[()]')
'\\[\\(\\)\\]'
```

See also:

- *escape_reg_exp()* (main definition)
- *escape_re()* (alias)

New in version 1.1.0.

pydash.strings.**escape_re**(*text*)

Escapes the RegExp special characters in *text*.

Parameters **text** (*str*) – String to escape.

Returns RegExp escaped string.

Return type str

Example

```
>>> escape_reg_exp('[()]')
'\\[\\(\\)\\]'
```

See also:

- *escape_reg_exp()* (main definition)
- *escape_re()* (alias)

New in version 1.1.0.

pydash.strings.**explode**(*text*, *separator=<pydash.helpers._NoValue object>*)

Splits *text* on *separator*. If *separator* not provided, then *text* is split on whitespace. If *separator* is falsey, then *text* is split on every character.

Parameters

- **text** (*str*) – String to explode.
- **separator** (*str, optional*) – Separator string to split on. Defaults to `NoValue`.

Returns Split string.

Return type list

Example

```
>>> split('one potato, two potatoes, three potatoes, four!')
['one', 'potato,', 'two', 'potatoes,', 'three', 'potatoes,', 'four!']
>>> split('one potato, two potatoes, three potatoes, four!', ',')
['one potato', ' two potatoes', ' three potatoes', ' four!']
```

See also:

- *split()* (main definition)
- *explode()* (alias)

New in version 2.0.0.

Changed in version 3.0.0: Changed *separator* default to `NoValue` and supported splitting on whitespace by default.

pydash.strings.**has_substr**(*text*, *subtext*)

Returns whether *subtext* is included in *text*.

Parameters

- **text** (*str*) – String to search.
- **subtext** (*str*) – String to search for.

Returns Whether *subtext* is found in *text*.

Return type bool

Example

```
>>> has_substr('abcdef', 'bc')
True
>>> has_substr('abcdef', 'bb')
False
```

New in version 3.0.0.

pydash.strings.**human_case**(*text*)

Converts *text* to human case which has only the first letter capitalized and each word separated by a space.

Parameters **text** (*str*) – String to convert.

Returns String converted to human case.

Return type str

Example

```
>>> human_case('abc-def_hij lmn')
'Abc def hij lmn'
>>> human_case('user_id')
'User'
```

New in version 3.0.0.

pydash.strings.**implode**(*array*, *separator=''*)

Joins an iterable into a string using *separator* between each element.

Parameters

- **array** (*iterable*) – Iterable to implode.
- **separator** (*str, optional*) – Separator to using when joining. Defaults to ''.

Returns Joined string.

Return type str

Example

```
>>> join(['a', 'b', 'c']) == 'abc'
True
>>> join([1, 2, 3, 4], '&') == '1&2&3&4'
True
>>> join('abcdef', '-') == 'a-b-c-d-e-f'
True
```

See also:

- *join()* (main definition)
- *implode()* (alias)

New in version 2.0.0.

Changed in version 3.0.0: Modified *implode()* to have *join()* as main definition and *implode()* as alias.

pydash.strings.**insert_substr**(*text*, *index*, *subtext*)
Insert *subtext* in *text* starting at position *index*.

Parameters

- **text** (*str*) – String to add substring to.
- **index** (*int*) – String index to insert into.
- **subtext** (*str*) – String to insert.

Returns Modified string.

Return type str

Example

```
>>> insert_substr('abcdef', 3, '--')
'abc--def'
```

New in version 3.0.0.

pydash.strings.**join**(*array*, *separator=''*)
Joins an iterable into a string using *separator* between each element.

Parameters

- **array** (*iterable*) – Iterable to implode.
- **separator** (*str, optional*) – Separator to using when joining. Defaults to `''`.

Returns Joined string.

Return type str

Example

```
>>> join(['a', 'b', 'c']) == 'abc'
True
>>> join([1, 2, 3, 4], '&') == '1&2&3&4'
True
>>> join('abcdef', '-') == 'a-b-c-d-e-f'
True
```

See also:

- *join()* (main definition)
- *implode()* (alias)

New in version 2.0.0.

Changed in version 3.0.0: Modified *implode()* to have *join()* as main definition and *implode()* as alias.

pydash.strings.**js_match**(*text*, *reg_exp*)
Return list of matches using Javascript style regular expression.

Parameters

- **text** (*str*) – String to evaluate.
- **reg_exp** (*str*) – Javascript style regular expression.

Returns List of matches.

Return type list

Example

```
>>> js_match('aaBBcc', '/bb/')
[]
>>> js_match('aaBBcc', '/bb/i')
['BB']
>>> js_match('aaBBccbb', '/bb/i')
['BB']
>>> js_match('aaBBccbb', '/bb/gi')
['BB', 'bb']
```

New in version 2.0.0.

Changed in version 3.0.0: Reordered arguments to make *text* first.

pydash.strings.**js_replace**(*text*, *reg_exp*, *repl*)

Replace *text* with *repl* using Javascript style regular expression to find matches.

Parameters

- **text** (*str*) – String to evaluate.
- **reg_exp** (*str*) – Javascript style regular expression.
- **repl** (*str*) – Replacement string.

Returns Modified string.

Return type str

Example

```
>>> js_replace('aaBBcc', '/bb/', 'X')
'aaBBcc'
>>> js_replace('aaBBcc', '/bb/i', 'X')
'aaXcc'
>>> js_replace('aaBBccbb', '/bb/i', 'X')
'aaXccbb'
>>> js_replace('aaBBccbb', '/bb/gi', 'X')
'aaXccX'
```

New in version 2.0.0.

Changed in version 3.0.0: Reordered arguments to make *text* first.

pydash.strings.**kebab_case**(*text*)

Converts *text* to kebab case (a.k.a. spinal case).

Parameters **text** (*str*) – String to convert.

Returns String converted to kebab case.

Return type str

Example

```
>>> kebab_case('a b c_d-e!f')
'a-b-c-d-e-f'
```

New in version 1.1.0.

pydash.strings.**lines**(*text*)
Split lines in *text* into an array.

Parameters **text** (*str*) – String to split.

Returns String split by lines.

Return type list

Example

```
>>> lines('a\nb\r\nc')
['a', 'b', 'c']
```

New in version 3.0.0.

pydash.strings.**number_format**(*number*, *scale=0*, *decimal_separator='.'*, *order_separator=''*)
Format a number to scale with custom decimal and order separators.

Parameters

- **number** (*int|float*) – Number to format.
- **scale** (*int, optional*) – Number of decimals to include. Defaults to `0`.
- **decimal_separator** (*str, optional*) – Decimal separator to use. Defaults to `'.'`.
- **order_separator** (*str, optional*) – Order separator to use. Defaults to `','`.

Returns Formatted number as string.

Return type str

Example

```
>>> number_format(1234.5678)
'1,235'
>>> number_format(1234.5678, 2, ',', '.')
'1.234,57'
```

New in version 3.0.0.

pydash.strings.**pad**(*text*, *length*, *chars=' '*)
Pads *text* on the left and right sides if it is shorter than the given padding length. The *chars* string may be truncated if the number of padding characters can't be evenly divided by the padding length.

Parameters

- **text** (*str*) – String to pad.
- **length** (*int*) – Amount to pad.
- **chars** (*str, optional*) – Characters to pad with. Defaults to `" "`.

Returns Padded string.

Return type str

Example

```
>>> pad('abc', 5)
' abc '
>>> pad('abc', 6, 'x')
'xabcxx'
>>> pad('abc', 5, '...')
'.abc.'
```

New in version 1.1.0.

Changed in version 3.0.0: Fix handling of multiple *chars* so that padded string isn't over padded.

pydash.strings.**pad_left**(*text*, *length*, *chars=' '*)

Pads *text* on the left side if it is shorter than the given padding length. The *chars* string may be truncated if the number of padding characters can't be evenly divided by the padding length.

Parameters

- **text** (*str*) – String to pad.
- **length** (*int*) – Amount to pad.
- **chars** (*str, optional*) – Characters to pad with. Defaults to `" "`.

Returns Padded string.

Return type str

Example

```
>>> pad_left('abc', 5)
'  abc'
>>> pad_left('abc', 5, '.')
'..abc'
```

New in version 1.1.0.

pydash.strings.**pad_right**(*text*, *length*, *chars=' '*)

Pads *text* on the right side if it is shorter than the given padding length. The *chars* string may be truncated if the number of padding characters can't be evenly divided by the padding length.

Parameters

- **text** (*str*) – String to pad.
- **length** (*int*) – Amount to pad.
- **chars** (*str, optional*) – Characters to pad with. Defaults to `" "`.

Returns Padded string.

Return type str

Example

```
>>> pad_right('abc', 5)
'abc  '
>>> pad_right('abc', 5, '.')
'abc..'
```

New in version 1.1.0.

pydash.strings.**pascal_case**(*text*, *strict=True*)

Like *camel_case()* except the first letter is capitalized.

Parameters

- **text** (*str*) – String to convert.
- **strict** (*bool, optional*) – Whether to cast rest of string to lower case. Defaults to `True`.

Returns String converted to class case.

Return type str

Example

```
>>> pascal_case('FOO BAR_bAz')
'FooBarBaz'
>>> pascal_case('FOO BAR_bAz', False)
'FooBarBAz'
```

New in version 3.0.0.

pydash.strings.**predecessor**(*char*)

Return the predecessor character of *char*.

Parameters **char** (*str*) – Character to find the predecessor of.

Returns Predecessor character.

Return type str

Example

```
>>> predecessor('c')
'b'
>>> predecessor('C')
'B'
>>> predecessor('3')
'2'
```

New in version 3.0.0.

pydash.strings.**prune**(*text*, *length=0*, *omission='...'*)

Like *truncate()* except it ensures that the pruned string doesn't exceed the original length, i.e., it avoids half-chopped words when truncating. If the pruned text + *omission* text is longer than the original text, then the original text is returned.

Parameters

- **text** (*str*) – String to prune.

- **length** (*int, optional*) – Target prune length. Defaults to `0`.
- **omission** (*str, optional*) – Omission text to append to the end of the pruned string. Defaults to `'...'`.

Returns Pruned string.

Return type str

Example

```
>>> prune('Fe fi fo fum', 5)
'Fe fi...'
>>> prune('Fe fi fo fum', 6)
'Fe fi...'
>>> prune('Fe fi fo fum', 7)
'Fe fi...'
>>> prune('Fe fi fo fum', 8, ',,,')
'Fe fi fo,,,'
```

New in version 3.0.0.

pydash.strings.**quote**(*text*, *quote_char=""*)

Quote a string with another string.

Parameters

- **text** (*str*) – String to be quoted.
- **quote_char** (*str, optional*) – the quote character. Defaults to `"`.

Returns the quoted string.

Return type str

Example

```
>>> quote('To be or not to be')
'"To be or not to be"'
>>> quote('To be or not to be', "'")
"'To be or not to be'"
```

New in version 2.4.0.

pydash.strings.**re_replace**(*text*, *pattern*, *repl*, *ignore_case=False*, *count=0*)

Replace occurrences of regex *pattern* with *repl* in *text*. Optionally, ignore case when replacing. Optionally, set *count* to limit number of replacements.

Parameters

- **text** (*str*) – String to replace.
- **pattern** (*str*) – String pattern to find and replace.
- **repl** (*str*) – String to substitute *pattern* with.
- **ignore_clase** (*bool, optional*) – Whether to ignore case when replacing. Defaults to `False`.
- **count** (*int, optional*) – Maximum number of occurrences to replace. Defaults to `0` which replaces all.

Returns Replaced string.

Return type str

Example

```
>>> re_replace('aabbcc', 'b', 'X')
'aaXXcc'
>>> re_replace('aabbcc', 'B', 'X', ignore_case=True)
'aaXXcc'
>>> re_replace('aabbcc', 'b', 'X', count=1)
'aaXbcc'
>>> re_replace('aabbcc', '[ab]', 'X')
'XXXXcc'
```

New in version 3.0.0.

pydash.strings.**repeat**(*text*, *n=0*)

Repeats the given string *n* times.

Parameters

- **text** (*str*) – String to repeat.
- **n** (*int, optional*) – Number of times to repeat the string.

Returns Repeated string.

Return type str

Example

```
>>> repeat('.', 5)
'.....'
```

New in version 1.1.0.

pydash.strings.**replace**(*text*, *pattern*, *repl*, *ignore_case=False*, *count=0*, *escape=True*)

Replace occurrences of *pattern* with *repl* in *text*. Optionally, ignore case when replacing. Optionally, set *count* to limit number of replacements.

Parameters

- **text** (*str*) – String to replace.
- **pattern** (*str*) – String pattern to find and replace.
- **repl** (*str*) – String to substitute *pattern* with.
- **ignore_clase** (*bool, optional*) – Whether to ignore case when replacing. Defaults to `False`.
- **count** (*int, optional*) – Maximum number of occurrences to replace. Defaults to `0` which replaces all.
- **escape** (*bool, optional*) – Whether to escape *pattern* when searching. This is needed if a literal replacement is desired when *pattern* may contain special regular expression characters. Defaults to `True`.

Returns Replaced string.

Return type str

Example

```
>>> replace('aabbcc', 'b', 'X')
'aaXXcc'
>>> replace('aabbcc', 'B', 'X', ignore_case=True)
'aaXXcc'
>>> replace('aabbcc', 'b', 'X', count=1)
'aaXbcc'
>>> replace('aabbcc', '[ab]', 'X')
'aabbcc'
>>> replace('aabbcc', '[ab]', 'X', escape=False)
'XXXXcc'
```

New in version 3.0.0.

pydash.strings.**separator_case**(*text*, *separator*)

Splits *text* on words and joins with *separator*.

Parameters

- **text** (*str*) – String to convert.
- **separator** (*str*) – Separator to join words with.

Returns Converted string.

Return type str

Example

```
>>> separator_case('a!!b___c.d', '-')
'a-b-c-d'
```

New in version 3.0.0.

pydash.strings.**series_phrase**(*items*, *separator=', '*, *last_separator=' and '*, *serial=False*)

Join items into a grammatical series phrase, e.g., `"item1, item2, item3 and item4"`.

Parameters

- **items** (*list*) – List of string items to join.
- **separator** (*str, optional*) – Item separator. Defaults to `', '`.
- **last_separator** (*str, optional*) – Last item separator. Defaults to `' and '`.
- **serial** (*bool, optional*) – Whether to include *separator* with *last_separator* when number of items is greater than 2. Defaults to `False`.

Returns Joined string.

Return type str

Example

Example:

```
>>> series_phrase(['apples', 'bananas', 'peaches'])
'apples, bananas and peaches'
>>> series_phrase(['apples', 'bananas', 'peaches'], serial=True)
```

```
    'apples, bananas, and peaches'
    >>> series_phrase(['apples', 'bananas', 'peaches'], '; ', ', or ')
    'apples; bananas, or peaches'
```

New in version 3.0.0.

pydash.strings.**series_phrase_serial**(*items*, *separator=', '*, *last_separator=' and '*)

Join items into a grammatical series phrase using a serial separator, e.g., `"item1, item2, item3, and item4"`.

Parameters

- **items** (*list*) – List of string items to join.
- **separator** (*str, optional*) – Item separator. Defaults to `', '`.
- **last_separator** (*str, optional*) – Last item separator. Defaults to `' and '`.

Returns Joined string.

Return type str

Example

```
    >>> series_phrase_serial(['apples', 'bananas', 'peaches'])
    'apples, bananas, and peaches'
```

New in version 3.0.0.

pydash.strings.**slugify**(*text*, *separator='-'*)

Convert *text* into an ASCII slug which can be used safely in URLs. Incoming *text* is converted to unicode and noramlzied using the `NFKD` form. This results in some accented characters being converted to their ASCII "equivalent" (e.g. `é` is converted to `e`). Leading and trailing whitespace is trimmed and any remaining whitespace or other special characters without an ASCII equivalent are replaced with `-`.

Parameters

- **text** (*str*) – String to slugify.
- **separator** (*str, optional*) – Separator to use. Defaults to `'-'`.

Returns Slugified string.

Return type str

Example

```
    >>> slugify('This is a slug.') == 'this-is-a-slug'
    True
    >>> slugify('This is a slug.', '+') == 'this+is+a+slug'
    True
```

New in version 3.0.0.

pydash.strings.**snake_case**(*text*)

Converts *text* to snake case.

Parameters **text** (*str*) – String to convert.

Returns String converted to snake case.

Return type str

Example

```
>>> snake_case('This is Snake Case!')
'this_is_snake_case'
```

See also:

- *snake_case()* (main definition)
- *underscore_case()* (alias)

New in version 1.1.0.

pydash.strings.**split**(*text*, *separator=<pydash.helpers._NoValue object>*)

Splits *text* on *separator*. If *separator* not provided, then *text* is split on whitespace. If *separator* is falsey, then *text* is split on every character.

Parameters

- **text** (*str*) – String to explode.
- **separator** (*str, optional*) – Separator string to split on. Defaults to NoValue.

Returns Split string.

Return type list

Example

```
>>> split('one potato, two potatoes, three potatoes, four!')
['one', 'potato,', 'two', 'potatoes,', 'three', 'potatoes,', 'four!']
>>> split('one potato, two potatoes, three potatoes, four!', ',')
['one potato', ' two potatoes', ' three potatoes', ' four!']
```

See also:

- *split()* (main definition)
- *explode()* (alias)

New in version 2.0.0.

Changed in version 3.0.0: Changed *separator* default to NoValue and supported splitting on whitespace by default.

pydash.strings.**start_case**(*text*)

Convert *text* to start case.

Parameters **text** (*str*) – String to convert.

Returns String converted to start case.

Return type str

Example

```
>>> start_case("fooBar")
'Foo Bar'
```

New in version 3.1.0.

pydash.strings.**starts_with**(*text*, *target*, *position=0*)
Checks if *text* starts with a given target string.

Parameters

- **text** (*str*) – String to check.
- **target** (*str*) – String to check for.
- **position** (*int, optional*) – Position to search from. Defaults to beginning of *text*.

Returns Whether *text* starts with *target*.

Return type bool

Example

```
>>> starts_with('abcdef', 'a')
True
>>> starts_with('abcdef', 'b')
False
>>> starts_with('abcdef', 'a', 1)
False
```

New in version 1.1.0.

pydash.strings.**strip_tags**(*text*)
Removes all HTML tags from *text*.

Parameters **text** (*str*) – String to strip.

Returns String without HTML tags.

Return type str

Example

```
>>> strip_tags('<a href="#">Some link</a>')
'Some link'
```

New in version 3.0.0.

pydash.strings.**substr_left**(*text*, *subtext*)
Searches *text* from left-to-right for *subtext* and returns a substring consisting of the characters in *text* that are to the left of *subtext* or all string if no match found.

Parameters

- **text** (*str*) – String to partition.
- **subtext** (*str*) – String to search for.

Returns Substring to left of *subtext*.

Return type str

Example

```
>>> substr_left('abcdefcdg', 'cd')
'ab'
```

New in version 3.0.0.

pydash.strings.**substr_left_end**(*text*, *subtext*)

Searches *text* from right-to-left for *subtext* and returns a substring consisting of the characters in *text* that are to the left of *subtext* or all string if no match found.

Parameters

- **text** (*str*) – String to partition.
- **subtext** (*str*) – String to search for.

Returns Substring to left of *subtext*.

Return type str

Example

```
>>> substr_left_end('abcdefcdg', 'cd')
'abcdef'
```

New in version 3.0.0.

pydash.strings.**substr_right**(*text*, *subtext*)

Searches *text* from right-to-left for *subtext* and returns a substring consisting of the characters in *text* that are to the right of *subtext* or all string if no match found.

Parameters

- **text** (*str*) – String to partition.
- **subtext** (*str*) – String to search for.

Returns Substring to right of *subtext*.

Return type str

Example

```
>>> substr_right('abcdefcdg', 'cd')
'efcdg'
```

New in version 3.0.0.

pydash.strings.**substr_right_end**(*text*, *subtext*)

Searches *text* from left-to-right for *subtext* and returns a substring consisting of the characters in *text* that are to the right of *subtext* or all string if no match found.

Parameters

- **text** (*str*) – String to partition.
- **subtext** (*str*) – String to search for.

Returns Substring to right of *subtext*.

Return type str

Example

```
>>> substr_right_end('abcdefcdg', 'cd')
'g'
```

New in version 3.0.0.

pydash.strings.**successor**(*char*)
Return the successor character of *char*.

Parameters **char** (*str*) – Character to find the successor of.

Returns Successor character.

Return type str

Example

```
>>> successor('b')
'c'
>>> successor('B')
'C'
>>> successor('2')
'3'
```

New in version 3.0.0.

pydash.strings.**surround**(*text*, *wrapper*)
Surround a string with another string.

Parameters

- **text** (*str*) – String to surround with *wrapper*.
- **wrapper** (*str*) – String by which *text* is to be surrounded.

Returns Surrounded string.

Return type str

Example

```
>>> surround('abc', '"')
'"abc"'
>>> surround('abc', '!')
'!abc!'
```

New in version 2.4.0.

pydash.strings.**swap_case**(*text*)
Swap case of *text* characters.

Parameters **text** (*str*) – String to swap case.

Returns String with swapped case.

Return type str

Example

```
>>> swap_case('aBcDeF')
'AbCdEf'
```

New in version 3.0.0.

pydash.strings.**title_case**(*text*)
Convert *text* to title case.

Parameters **text** (*str*) – String to convert.

Returns String converted to title case.

Return type str

Example

```
>>> title_case("bob's shop")
"Bob's Shop"
```

New in version 3.0.0.

pydash.strings.**trim**(*text*, *chars=None*)
Removes leading and trailing whitespace or specified characters from *text*.

Parameters

- **text** (*str*) – String to trim.
- **chars** (*str, optional*) – Specific characters to remove.

Returns Trimmed string.

Return type str

Example

```
>>> trim('  abc efg\r\n ')
'abc efg'
```

New in version 1.1.0.

pydash.strings.**trim_left**(*text*, *chars=None*)
Removes leading whitespace or specified characters from *text*.

Parameters

- **text** (*str*) – String to trim.
- **chars** (*str, optional*) – Specific characters to remove.

Returns Trimmed string.

Return type str

Example

```
>>> trim_left('  abc efg\r\n ')
'abc efg\r\n '
```

New in version 1.1.0.

pydash.strings.**trim_right**(*text*, *chars=None*)

Removes trailing whitespace or specified characters from *text*.

Parameters

- **text** (*str*) – String to trim.
- **chars** (*str, optional*) – Specific characters to remove.

Returns Trimmed string.

Return type str

Example

```
>>> trim_right('  abc efg\r\n ')
'  abc efg'
```

New in version 1.1.0.

pydash.strings.**trunc**(*text*, *length=30*, *omission='...'*, *separator=None*)

Truncates *text* if it is longer than the given maximum string length. The last characters of the truncated string are replaced with the omission string which defaults to `...`.

Parameters

- **text** (*str*) – String to truncate.
- **length** (*int, optional*) – Maximum string length. Defaults to `30`.
- **omission** (*str, optional*) – String to indicate text is omitted.
- **separator** (*mixed, optional*) – Separator pattern to truncate to.

Returns Truncated string.

Return type str

Example

```
>>> truncate('hello world', 5)
'he...'
>>> truncate('hello world', 5, '..')
'hel..'
>>> truncate('hello world', 10)
'hello w...'
>>> truncate('hello world', 10, separator=' ')
'hello...'
```

See also:

- *truncate()* (main definition)

- *trunc()* (alias)

New in version 1.1.0.

Changed in version 3.0.0: Made *truncate()* main function definition and added *trunc()* as alias.

pydash.strings.**truncate**(*text*, *length=30*, *omission='...'*, *separator=None*)

Truncates *text* if it is longer than the given maximum string length. The last characters of the truncated string are replaced with the omission string which defaults to `...`.

Parameters

- **text** (*str*) – String to truncate.
- **length** (*int, optional*) – Maximum string length. Defaults to `30`.
- **omission** (*str, optional*) – String to indicate text is omitted.
- **separator** (*mixed, optional*) – Separator pattern to truncate to.

Returns Truncated string.

Return type str

Example

```
>>> truncate('hello world', 5)
'he...'
>>> truncate('hello world', 5, '..')
'hel..'
>>> truncate('hello world', 10)
'hello w...'
>>> truncate('hello world', 10, separator=' ')
'hello...'
```

See also:

- *truncate()* (main definition)
- *trunc()* (alias)

New in version 1.1.0.

Changed in version 3.0.0: Made *truncate()* main function definition and added *trunc()* as alias.

pydash.strings.**underscore_case**(*text*)

Converts *text* to snake case.

Parameters **text** (*str*) – String to convert.

Returns String converted to snake case.

Return type str

Example

```
>>> snake_case('This is Snake Case!')
'this_is_snake_case'
```

See also:

- *snake_case()* (main definition)
- *underscore_case()* (alias)

New in version 1.1.0.

pydash.strings.**unescape**(*text*)

The inverse of *escape()*. This method converts the HTML entities `&`, `<`, `>`, `&quot;`, `&#39;`, and ``` in *text* to their corresponding characters.

Parameters **text** (*str*) – String to unescape.

Returns HTML unescaped string.

Return type str

Example

```
>>> results = unescape('&quot;1 &gt; 2 &amp;&amp; 3 &lt; 4&quot;')
>>> results == '"1 > 2 && 3 < 4"'
True
```

New in version 1.0.0.

Changed in version 1.1.0: Moved to *pydash.strings*.

pydash.strings.**unquote**(*text*, *quote_char=""*)

Unquote *text* by removing *quote_char* if *text* begins and ends with it.

Parameters **text** (*str*) – String to unquote.

Returns Unquoted string.

Return type str

Example

```
>>> unquote('"abc"')
'abc'
>>> unquote('"abc"', '#')
'"abc"'
>>> unquote('#abc', '#')
'#abc'
>>> unquote('#abc#', '#')
'abc'
```

New in version 3.0.0.

pydash.strings.**url**(**paths*, ***params*)

Combines a series of URL paths into a single URL. Optionally, pass in keyword arguments to append query parameters.

Parameters **paths** (*str*) – URL paths to combine.

Keyword Arguments **params** (*str, optional*) – Query parameters.

Returns URL string.

Return type str

Example

```
>>> link = url('a', 'b', ['c', 'd'], '/', q='X', y='Z')
>>> path, params = link.split('?')
>>> path == 'a/b/c/d/'
True
>>> set(params.split('&')) == set(['q=X', 'y=Z'])
True
```

New in version 2.2.0.

pydash.strings.**words**(*text*, *pattern=None*)

Return list of words contained in *text*.

Parameters

- **text** (*str*) – String to split.
- **pattern** (*str, optional*) – Custom pattern to split words on. Defaults to None.

Returns List of words.

Return type list

Example

```
>>> words('a b, c; d-e')
['a', 'b', 'c', 'd', 'e']
>>> words('fred, barney, & pebbles', '/[^, ]+/g')
['fred', 'barney', '&', 'pebbles']
```

New in version 2.0.0.

Changed in version 3.2.0: Added *pattern* argument.

Changed in version 3.2.0: Improved matching for one character words.

4.1.10 Utilities

Utility functions.

New in version 1.0.0.

pydash.utilities.**attempt**(*func*, **args*, ***kargs*)

Attempts to execute *func*, returning either the result or the caught error object.

Parameters **func** (*function*) – The function to attempt.

Returns Returns the *func* result or error object.

Return type mixed

Example

```
>>> results = attempt(lambda x: x/0, 1)
>>> assert isinstance(results, ZeroDivisionError)
```

New in version 1.1.0.

pydash.utilities.**constant**(*value*)

Creates a function that returns *value*.

Parameters **value** (*mixed*) – Constant value to return.

Returns Function that always returns *value*.

Return type function

Example

```
>>> pi = constant(3.14)
>>> pi() == 3.14
True
```

New in version 1.0.0.

pydash.utilities.**callback**(*func*)

Return a pydash style callback. If *func* is a property name the created callback will return the property value for a given element. If *func* is an object the created callback will return `True` for elements that contain the equivalent object properties, otherwise it will return `False`.

Parameters **func** (*mixed*) – Object to create callback function from.

Returns Callback function.

Return type function

Example

```
>>> get_data = iteratee('data')
>>> get_data({'data': [1, 2, 3]})
[1, 2, 3]
>>> is_active = iteratee({'active': True})
>>> is_active({'active': True})
True
>>> is_active({'active': 0})
False
>>> iteratee(['a', 5])({'a': 5})
True
>>> iteratee(['a.b'])({'a.b': 5})
5
>>> iteratee('a.b')({'a': {'b': 5}})
5
>>> iteratee(lambda a, b: a + b)(1, 2)
3
>>> ident = iteratee(None)
>>> ident('a')
'a'
>>> ident(1, 2, 3)
1
```

See also:

- *iteratee()* (main definition)
- *callback()* (alias)

New in version 1.0.0.

Changed in version 2.0.0: Renamed `create_callback()` to *iteratee()*.

Changed in version 3.0.0: Made pluck style callback support deep property access.

Changed in version 3.1.0: - Added support for shallow pluck style property access via single item list/tuple. - Added support for matches property style callback via two item list/tuple.

pydash.utilities.**deep_property**(*path*)

Creates a *pydash.collections.pluck()* style function, which returns the key value of a given object.

Parameters **key** (*mixed*) – Key value to fetch from object.

Returns Function that returns object's key value.

Return type function

Example

```
>>> deep_property('a.b.c')({'a': {'b': {'c': 1}}})
1
>>> deep_property('a.1.0.b')({'a': [5, [{'b': 1}]]})
1
>>> deep_property('a.1.0.b')({}) is None
True
```

See also:

- *deep_property()* (main definition)
- *deep_prop()* (alias)

New in version 1.0.0.

pydash.utilities.**deep_prop**(*path*)

Creates a *pydash.collections.pluck()* style function, which returns the key value of a given object.

Parameters **key** (*mixed*) – Key value to fetch from object.

Returns Function that returns object's key value.

Return type function

Example

```
>>> deep_property('a.b.c')({'a': {'b': {'c': 1}}})
1
>>> deep_property('a.1.0.b')({'a': [5, [{'b': 1}]]})
1
>>> deep_property('a.1.0.b')({}) is None
True
```

See also:

- *deep_property()* (main definition)
- *deep_prop()* (alias)

New in version 1.0.0.

pydash.utilities.**identity**(**args*)

Return the first argument provided to it.

Parameters ***args** (*mixed*) – Arguments.

Returns First argument or `None`.

Return type mixed

Example

```
>>> identity(1)
1
>>> identity(1, 2, 3)
1
>>> identity() is None
True
```

New in version 1.0.0.

pydash.utilities.**iteratee**(*func*)

Return a pydash style callback. If *func* is a property name the created callback will return the property value for a given element. If *func* is an object the created callback will return `True` for elements that contain the equivalent object properties, otherwise it will return `False`.

Parameters **func** (*mixed*) – Object to create callback function from.

Returns Callback function.

Return type function

Example

```
>>> get_data = iteratee('data')
>>> get_data({'data': [1, 2, 3]})
[1, 2, 3]
>>> is_active = iteratee({'active': True})
>>> is_active({'active': True})
True
>>> is_active({'active': 0})
False
>>> iteratee(['a', 5])({'a': 5})
True
>>> iteratee(['a.b'])({'a.b': 5})
5
>>> iteratee('a.b')({'a': {'b': 5}})
5
>>> iteratee(lambda a, b: a + b)(1, 2)
3
>>> ident = iteratee(None)
>>> ident('a')
'a'
>>> ident(1, 2, 3)
1
```

See also:

- *iteratee()* (main definition)

- *callback()* (alias)

New in version 1.0.0.

Changed in version 2.0.0: Renamed `create_callback()` to *iteratee()*.

Changed in version 3.0.0: Made pluck style callback support deep property access.

Changed in version 3.1.0: - Added support for shallow pluck style property access via single item list/tuple. - Added support for matches property style callback via two item list/tuple.

pydash.utilities.**matches**(*source*)

Creates a *pydash.collections.where()* style predicate function which performs a deep comparison between a given object and the *source* object, returning `True` if the given object has equivalent property values, else `False`.

Parameters **source** (*dict*) – Source object used for comparision.

Returns Function that compares an object to *source* and returns whether the two objects contain the same items.

Return type function

Example

```
>>> matches({'a': {'b': 2}})({'a': {'b': 2, 'c':3}})
True
>>> matches({'a': 1})({'b': 2, 'a': 1})
True
>>> matches({'a': 1})({'b': 2, 'a': 2})
False
```

New in version 1.0.0.

Changed in version 3.0.0: Use *pydash.predicates.is_match()* as matching function.

pydash.utilities.**matches_property**(*key*, *value*)

Creates a function that compares the property value of *key* on a given object to *value*.

Parameters

- **key** (*str*) – Object key to match against.
- **value** (*mixed*) – Value to compare to.

Returns Function that compares *value* to an object's *key* and returns whether they are equal.

Return type function

Example

```
>>> matches_property('a', 1)({'a': 1, 'b': 2})
True
>>> matches_property(0, 1)([1, 2, 3])
True
>>> matches_property('a', 2)({'a': 1, 'b': 2})
False
```

New in version 3.1.0.

pydash.utilities.**memoize**(*func*, *resolver=None*)

Creates a function that memoizes the result of *func*. If *resolver* is provided it will be used to determine the cache key for storing the result based on the arguments provided to the memoized function. By default, all arguments provided to the memoized function are used as the cache key. The result cache is exposed as the cache property on the memoized function.

Parameters

- **func** (*function*) – Function to memoize.
- **resolver** (*function, optional*) – Function that returns the cache key to use.

Returns Memoized function.

Return type function

Example

```
>>> ident = memoize(identity)
>>> ident(1)
1
>>> ident.cache['(1,){}'] == 1
True
>>> ident(1, 2, 3)
1
>>> ident.cache['(1, 2, 3){}'] == 1
True
```

New in version 1.0.0.

pydash.utilities.**method**(*path*, **args*, ***kargs*)

Creates a function that invokes the method at *path* on a given object. Any additional arguments are provided to the invoked method.

Parameters

- **path** (*str*) – Object path of method to invoke.
- ***args** (*mixed*) – Global arguments to apply to method when invoked.
- ****kargs** (*mixed*) – Global keyword argument to apply to method when invoked.

Returns Function that invokes method located at path for object.

Return type function

Example

```
>>> obj = {'a': {'b': [None, lambda x: x]}}
>>> echo = method('a.b.1')
>>> echo(obj, 1) == 1
True
>>> echo(obj, 'one') == 'one'
True
```

New in version 3.3.0.

pydash.utilities.**method_of**(*obj*, **args*, ***kargs*)

The opposite of *method()*. This method creates a function that invokes the method at a given path on object. Any additional arguments are provided to the invoked method.

Parameters

- **obj** (*mixed*) – The object to query.
- ***args** (*mixed*) – Global arguments to apply to method when invoked.
- ****kargs** (*mixed*) – Global keyword argument to apply to method when invoked.

Returns Function that invokes method located at path for object.

Return type function

Example

```
>>> obj = {'a': {'b': [None, lambda x: x]}}
>>> dispatch = method_of(obj)
>>> dispatch('a.b.1', 1) == 1
True
>>> dispatch('a.b.1', 'one') == 'one'
True
```

New in version 3.3.0.

pydash.utilities.**noop**(**args*, ***kargs*)

A no-operation function.

New in version 1.0.0.

pydash.utilities.**now**()

Return the number of milliseconds that have elapsed since the Unix epoch (1 January 1970 00:00:00 UTC).

Returns Milliseconds since Unix epoch.

Return type int

New in version 1.0.0.

Changed in version 3.0.0: Use `datetime` module for calculating elapsed time.

pydash.utilities.**prop**(*key*)

Creates a *pydash.collections.pluck()* style function, which returns the key value of a given object.

Parameters **key** (*mixed*) – Key value to fetch from object.

Returns Function that returns object's key value.

Return type function

Example

```
>>> get_data = prop('data')
>>> get_data({'data': 1})
1
>>> get_data({}) is None
True
>>> get_first = prop(0)
>>> get_first([1, 2, 3])
1
```

See also:

- *property_()* (main definition)

- *prop()* (alias)

New in version 1.0.0.

pydash.utilities.**prop_of**(*obj*)

The inverse of *property_()*. This method creates a function that returns the key value of a given key on *obj*.

Parameters **obj** (*dict | list*) – Object to fetch values from.

Returns Function that returns object's key value.

Return type function

Example

```
>>> getter = prop_of({'a': 1, 'b': 2, 'c': 3})
>>> getter('a')
1
>>> getter('b')
2
>>> getter('x') is None
True
```

See also:

- *property_of()* (main definition)
- *prop_of()* (alias)

New in version 3.0.0.

pydash.utilities.**property_**(*key*)

Creates a *pydash.collections.pluck()* style function, which returns the key value of a given object.

Parameters **key** (*mixed*) – Key value to fetch from object.

Returns Function that returns object's key value.

Return type function

Example

```
>>> get_data = prop('data')
>>> get_data({'data': 1})
1
>>> get_data({}) is None
True
>>> get_first = prop(0)
>>> get_first([1, 2, 3])
1
```

See also:

- *property_()* (main definition)
- *prop()* (alias)

New in version 1.0.0.

pydash.utilities.**property_of**(*obj*)

The inverse of *property_()*. This method creates a function that returns the key value of a given key on *obj*.

Parameters **obj** (*dict | list*) – Object to fetch values from.

Returns Function that returns object's key value.

Return type function

Example

```
>>> getter = prop_of({'a': 1, 'b': 2, 'c': 3})
>>> getter('a')
1
>>> getter('b')
2
>>> getter('x') is None
True
```

See also:

- *property_of()* (main definition)
- *prop_of()* (alias)

New in version 3.0.0.

pydash.utilities.**random**(*start=0*, *stop=1*, *floating=False*)

Produces a random number between *start* and *stop* (inclusive). If only one argument is provided a number between 0 and the given number will be returned. If floating is truthy or either *start* or *stop* are floats a floating-point number will be returned instead of an integer.

Parameters

- **start** (*int*) – Minimum value.
- **stop** (*int*) – Maximum value.
- **floating** (*bool, optional*) – Whether to force random value to `float`. Default is `False`.

Returns Random value.

Return type int|float

Example

```
>>> 0 <= random() <= 1
True
>>> 5 <= random(5, 10) <= 10
True
>>> isinstance(random(floating=True), float)
True
```

New in version 1.0.0.

pydash.utilities.**range_**(**args*)

Creates a list of numbers (positive and/or negative) progressing from start up to but not including end. If start is less than stop a zero-length range is created unless a negative step is specified.

Parameters

- **stop** (*int*) – Integer - 1 to stop at. Defaults to `1`.
- **start** (*int, optional*) – Integer to start with. Defaults to `0`.
- **step** (*int, optional*) – If positive the last element is the largest `start + i * step` less than *stop*. If negative the last element is the smallest `start + i * step` greater than *stop*. Defaults to `1`.

Returns List of integers in range

Return type list

Example

```
>>> list(range_(5))
[0, 1, 2, 3, 4]
>>> list(range_(1, 4))
[1, 2, 3]
>>> list(range_(0, 6, 2))
[0, 2, 4]
```

New in version 1.0.0.

Changed in version 1.1.0: Moved to `pydash.uilities`.

Changed in version 3.0.0: Return generator instead of list.

pydash.utilities.**result**(*obj*, *key*, *default=None*)

Return the value of property *key* on *obj*. If *key* value is a function it will be invoked and its result returned, else the property value is returned. If *obj* is falsey then *default* is returned.

Parameters

- **obj** (*list|dict*) – Object to retrieve result from.
- **key** (*mixed*) – Key or index to get result from.
- **default** (*mixed, optional*) – Default value to return if *obj* is falsey. Defaults to `None`.

Returns Result of `obj[key]` or `None`.

Return type mixed

Example

```
>>> result({'a': 1, 'b': lambda: 2}, 'a')
1
>>> result({'a': 1, 'b': lambda: 2}, 'b')
2
>>> result({'a': 1, 'b': lambda: 2}, 'c') is None
True
>>> result({'a': 1, 'b': lambda: 2}, 'c', default=False)
False
```

New in version 1.0.0.

Changed in version 2.0.0: Added `default` argument.

pydash.utilities.**times**(*callback*, *n*)

Executes the callback *n* times, returning a list of the results of each callback execution. The callback is invoked with one argument: `(index)`.

Parameters

- **callback** (*function*) – Function to execute.
- **n** (*int*) – Number of times to execute *callback*.

Returns A list of results from calling *callback*.

Return type list

Example

```
>>> times(lambda i: i, 5)
[0, 1, 2, 3, 4]
```

New in version 1.0.0.

Changed in version 3.0.0: Reordered arguments to make *callback* first.

pydash.utilities.**unique_id**(*prefix=None*)

Generates a unique ID. If *prefix* is provided the ID will be appended to it.

Parameters **prefix** (*str, optional*) – String prefix to prepend to ID value.

Returns ID value.

Return type str

Example

```
>>> unique_id()
'1'
>>> unique_id('id_')
'id_2'
>>> unique_id()
'3'
```

New in version 1.0.0.

CHAPTER 5

Project Info

5.1 License

5.2 Versioning

This project follows Semantic Versioning with the following caveats:

- Only the public API (i.e. the objects imported into the `pydash` module) will maintain backwards compatibility between MINOR version bumps.
- Objects within any other parts of the library are not guaranteed to not break between MINOR version bumps.

With that in mind, it is recommended to only use or import objects from the main module, `pydash`.

5.3 Changelog

5.3.1 v3.4.8 (2017-01-05)

- Make internal function inspection methods work with Python 3 annotations. Thanks tgriesser!

5.3.2 v3.4.7 (2016-11-01)

- Fix bug in `get` where an iterable default was iterated over instead of being returned when an object path wasn't found. Thanks urbnjamesmi1!

5.3.3 v3.4.6 (2016-10-31)

- Fix bug in `get` where casting a string key to integer resulted in an uncaught exception instead of the default value being returned instead. Thanks urbnjamesmi1!

5.3.4 v3.4.5 (2016-10-16)

- Add optional `default` parameter to `min_` and `max_` functions that is used when provided iterable is empty.
- Fix bug in `is_match` where comparison between an empty `source` argument returned `None` instead of `True`.

5.3.5 v3.4.4 (2016-09-06)

- Shallow copy each source in `assign`/`extend` instead of deep copying.
- Call `copy.deepcopy` in `merge` instead of the more resource intensive `clone_deep`.

5.3.6 v3.4.3 (2016-04-07)

- Fix minor issue in deep path string parsing so that list indexing in paths can be specified as `foo[0][1].bar` instead of `foo.[0].[1].bar`. Both formats are now supported.

5.3.7 v3.4.2 (2016-03-24)

- Fix bug in `start_case` where capitalized characters after the first character of a word where mistakenly cast to lower case.

5.3.8 v3.4.1 (2015-11-03)

- Fix Python 3.5, inspect, and pytest compatibility issue with `py_` chaining object when doctest run on `pydash.__init__.py`.

5.3.9 v3.4.0 (2015-09-22)

- Optimize callback system for performance.
  - Explicitly store arg count on callback for `pydash` generated callbacks where the arg count is known. This avoids the costly `inspect.getargspec` call.
  - Eliminate usage of costly `guess_builtin_argcount` which parsed docstrings, and instead only ever pass a single argument to a builtin callback function.
- Optimize `get`/`set` so that regex parsing is only done when special characters are contained in the path key whereas before, all string paths were parsed.

- Optimize `is_builtin` by checking for `BuiltinFunctionType` instance and then using `dict` look up table instead of a `list` look up.
- Optimize `is_match` by replacing call to `has` with a `try/except` block.
- Optimize `push`/`append` by using a native loop instead of callback mapping.

5.3.10 v3.3.0 (2015-07-23)

- Add `ceil`.
- Add `defaults_deep`.
- Add `floor`.
- Add `get`.
- Add `gt`.
- Add `gte`.
- Add `is_iterable`.
- Add `lt`.
- Add `lte`.
- Add `map_keys`.
- Add `method`.
- Add `method_of`.
- Add `mod_args`.
- Add `set_`.
- Add `unzip_with`.
- Add `zip_with`.
- Make `add` support adding two numbers if passed in positionally.
- Make `get` main definition and `get_path` its alias.
- Make `set_` main definition and `deep_set` its alias.

5.3.11 v3.2.2 (2015-04-29)

- Catch `AttributeError` in `helpers.get_item` and return default value if set.

5.3.12 v3.2.1 (2015-04-29)

- Fix bug in `reduce_right` where collection was not reversed correctly.

5.3.13 v3.2.0 (2015-03-03)

- Add `sort_by_order` as alias of `sort_by_all`.
- Fix `is_match` to not compare `obj` and `source` types using `type` and instead use `isinstance` comparisons exclusively.
- Make `sort_by_all` accept an `orders` argument for specifying the sort order of each key via boolean `True` (for ascending) and `False` (for descending).
- Make `words` accept a `pattern` argument to override the default regex used for splitting words.
- Make `words` handle single character words better.

5.3.14 v3.1.0 (2015-02-28)

- Add `fill`.
- Add `in_range`.
- Add `matches_property`.
- Add `spread`.
- Add `start_case`.
- Make callbacks support `matches_property` style as `[key, value]` or `(key, value)`.
- Make callbacks support shallow `property` style callbacks as `[key]` or `(key,)`.

5.3.15 v3.0.0 (2015-02-25)

- Add `ary`.
- Add `chars`.
- Add `chop`.
- Add `chop_right`.
- Add `clean`.
- Add `commit` method to `chain` that returns a new chain with the computed `chain.value()` as the initial value of the chain.
- Add `count_substr`.
- Add `decapitalize`.
- Add `duplicates`.
- Add `has_substr`.
- Add `human_case`.
- Add `insert_substr`.
- Add `is_blank`.
- Add `is_bool` as alias of `is_boolean`.
- Add `is_builtin`, `is_native`.
- Add `is_dict` as alias of `is_plain_object`.

- Add `is_int` as alias of `is_integer`.
- Add `is_match`.
- Add `is_num` as alias of `is_number`.
- Add `is_tuple`.
- Add `join` as alias of `implode`.
- Add `lines`.
- Add `number_format`.
- Add `pascal_case`.
- Add `plant` method to `chain` that returns a cloned chain with a new initial value.
- Add `predecessor`.
- Add `property_of`, `prop_of`.
- Add `prune`.
- Add `re_replace`.
- Add `rearg`.
- Add `replace`.
- Add `run` as alias of `chain.value`.
- Add `separator_case`.
- Add `series_phrase`.
- Add `series_phrase_serial`.
- Add `slugify`.
- Add `sort_by_all`.
- Add `strip_tags`.
- Add `substr_left`.
- Add `substr_left_end`.
- Add `substr_right`.
- Add `substr_right_end`.
- Add `successor`.
- Add `swap_case`.
- Add `title_case`.
- Add `truncate` as alias of `trunc`.
- Add `to_boolean`.
- Add `to_dict`, `to_plain_object`.
- Add `to_number`.
- Add `underscore_case` as alias of `snake_case`.
- Add `unquote`.
- Fix `deep_has` to return `False` when `ValueError` raised during path checking.

- Fix `pad` so that it doesn't over pad beyond provided length.
- Fix `trunc`/`truncate` so that they handle texts shorter than the max string length correctly.
- Make the following functions work with empty strings and `None`: (**breaking change**) Thanks k7sleeper!
  - `camel_case`
  - `capitalize`
  - `chars`
  - `chop`
  - `chop_right`
  - `class_case`
  - `clean`
  - `count_substr`
  - `decapitalize`
  - `ends_with`
  - `join`
  - `js_replace`
  - `kebab_case`
  - `lines`
  - `quote`
  - `re_replace`
  - `replace`
  - `series_phrase`
  - `series_phrase_serial`
  - `starts_with`
  - `surround`
- Make callback invocation have better support for builtin functions and methods. Previously, if one wanted to pass a builtin function or method as a callback, it had to be wrapped in a lambda which limited the number of arguments that would be passed it. For example, `_.each([1, 2, 3], array.append)` would fail and would need to be converted to `_.each([1, 2, 3], lambda item: array.append(item)`. That is no longer the case as the non-wrapped method is now supported.
- Make `capitalize` accept `strict` argument to control whether to convert the rest of the string to lower case or not. Defaults to `True`.
- Make `chain` support late passing of initial `value` argument.
- Make `chain` not store computed `value()`. (**breaking change**)
- Make `drop`, `drop_right`, `take`, and `take_right` have default `n=1`.
- Make `is_indexed` return `True` for tuples.
- Make `partial` and `partial_right` accept keyword arguments.
- Make `pluck` style callbacks support deep paths. (**breaking change**)
- Make `re_replace` accept non-string arguments.

- Make `sort_by` accept `reverse` parameter.
- Make `splice` work with strings.
- Make `to_string` convert `None` to empty string. (**breaking change**)
- Move `arrays.join` to `strings.join`. (**breaking change**)
- Rename `join`/`implode`'s second parameter from `delimiter` to `separator`. (**breaking change**)
- Rename `split`/`explode`'s second parameter from `delimiter` to `separator`. (**breaking change**)
- Reorder function arguments for `after` from `(n, func)` to `(func, n)`. (**breaking change**)
- Reorder function arguments for `before` from `(n, func)` to `(func, n)`. (**breaking change**)
- Reorder function arguments for `times` from `(n, callback)` to `(callback, n)`. (**breaking change**)
- Reorder function arguments for `js_match` from `(reg_exp, text)` to `(text, reg_exp)`. (**breaking change**)
- Reorder function arguments for `js_replace` from `(reg_exp, text, repl)` to `(text, reg_exp, repl)`. (**breaking change**)
- Support iteration over class instance properties for non-list, non-dict, and non-iterable objects.

5.3.16 v2.4.2 (2015-02-03)

- Fix `remove` so that array is modified after callback iteration.

5.3.17 v2.4.1 (2015-01-11)

- Fix `kebab_case` so that it casts string to lower case.

5.3.18 v2.4.0 (2015-01-07)

- Add `ensure_ends_with`. Thanks k7sleeper!
- Add `ensure_starts_with`. Thanks k7sleeper!
- Add `quote`. Thanks k7sleeper!
- Add `surround`. Thanks k7sleeper!

5.3.19 v2.3.2 (2014-12-10)

- Fix `merge` and `assign`/`extend` so they apply `clone_deep` to source values before assigning to destination object.
- Make `merge` accept a callback as a positional argument if it is last.

5.3.20 v2.3.1 (2014-12-07)

- Add `pipe` and `pipe_right` as aliases of `flow` and `flow_right`.
- Fix `merge` so that trailing `{}` or `[]` don't overwrite previous source values.
- Make `py_` an alias for `_`.

5.3.21 v2.3.0 (2014-11-10)

- Support `type` callbacks (e.g. `int`, `float`, `str`, etc.) by only passing a single callback argument when invoking the callback.
- Drop official support for Python 3.2. Too many testing dependencies no longer work on it.

5.3.22 v2.2.0 (2014-10-28)

- Add `append`.
- Add `deep_get`.
- Add `deep_has`.
- Add `deep_map_values`.
- Add `deep_set`.
- Add `deep_pluck`.
- Add `deep_property`.
- Add `join`.
- Add `pop`.
- Add `push`.
- Add `reverse`.
- Add `shift`.
- Add `sort`.
- Add `splice`.
- Add `unshift`.
- Add `url`.
- Fix bug in `snake_case` that resulted in returned string not being converted to lower case.
- Fix bug in chaining method access test which skipped the actual test.
- Make _ instance alias method access to methods with a trailing underscore in their name. For example, `_.map()` becomes an alias for `map_()`.
- Make `deep_prop` an alias of `deep_property`.
- Make `has` work with deep paths.
- Make `has_path` an alias of `deep_has`.
- Make `get_path` handle escaping the `.` delimiter for string keys.
- Make `get_path` handle list indexing using strings such as `'0.1.2'` to access `'value'` in `[[0, [0, 0, 'value']]]`.
- Make `concat` an alias of `cat`.

5.3.23 v2.1.0 (2014-09-17)

- Add `add`, `sum_`.
- Add `average`, `avg`, `mean`.
- Add `mapiter`.
- Add `median`.
- Add `moving_average`, `moving_avg`.
- Add `power`, `pow_`.
- Add `round_`, `curve`.
- Add `scale`.
- Add `slope`.
- Add `std_deviation`, `sigma`.
- Add `transpose`.
- Add `variance`.
- Add `zscore`.

5.3.24 v2.0.0 (2014-09-11)

- Add _ instance that supports both method chaining and module method calling.
- Add `cat`.
- Add `conjoin`.
- Add `deburr`.
- Add `disjoin`.
- Add `explode`.
- Add `flatten_deep`.
- Add `flow`.
- Add `flow_right`.
- Add `get_path`.
- Add `has_path`.
- Add `implode`.
- Add `intercalate`.
- Add `interleave`.
- Add `intersperse`.
- Add `is_associative`.
- Add `is_even`.
- Add `is_float`.
- Add `is_decreasing`.

- Add `is_increasing`.
- Add `is_indexed`.
- Add `is_instance_of`.
- Add `is_integer`.
- Add `is_json`.
- Add `is_monotone`.
- Add `is_negative`.
- Add `is_odd`.
- Add `is_positive`.
- Add `is_strictly_decreasing`.
- Add `is_strictly_increasing`.
- Add `is_zero`.
- Add `iterated`.
- Add `js_match`.
- Add `js_replace`.
- Add `juxtapose`.
- Add `mapcat`.
- Add `reductions`.
- Add `reductions_right`.
- Add `rename_keys`.
- Add `set_path`.
- Add `split_at`.
- Add `thru`.
- Add `to_string`.
- Add `update_path`.
- Add `words`.
- Make callback function calling adapt to argspec of given callback function. If, for example, the full callback signature is `(item, index, obj)` but the passed in callback only supports `(item)`, then only `item` will be passed in when callback is invoked. Previously, callbacks had to support all arguments or implement star-args.
- Make `chain` lazy and only compute the final value when `value` called.
- Make `compose` an alias of `flow_right`.
- Make `flatten` shallow by default, remove callback option, and add `is_deep` option. (**breaking change**)
- Make `is_number` return `False` for boolean `True` and `False`. (**breaking change**)
- Make `invert` accept `multivalue` argument.
- Make `result` accept `default` argument.
- Make `slice_` accept optional `start` and `end` arguments.
- Move files in `pydash/api/` to `pydash/`. (**breaking change**)

- Move predicate functions from `pydash.api.objects` to `pydash.api.predicates`. (**breaking change**)
- Rename `create_callback` to `iteratee`. (**breaking change**)
- Rename `functions` to `callables` in order to allow `functions.py` to exist at the root of the pydash module folder. (**breaking change**)
- Rename *private* utility function `_iter_callback` to `itercallback`. (**breaking change**)
- Rename *private* utility function `_iter_list_callback` to `iterlist_callback`. (**breaking change**)
- Rename *private* utility function `_iter_dict_callback` to `iterdict_callback`. (**breaking change**)
- Rename *private* utility function `_iterate` to `iterator`. (**breaking change**)
- Rename *private* utility function `_iter_dict` to `iterdict`. (**breaking change**)
- Rename *private* utility function `_iter_list` to `iterlist`. (**breaking change**)
- Rename *private* utility function `_iter_unique` to `iterunique`. (**breaking change**)
- Rename *private* utility function `_get_item` to `getitem`. (**breaking change**)
- Rename *private* utility function `_set_item` to `setitem`. (**breaking change**)
- Rename *private* utility function `_deprecated` to `deprecated`. (**breaking change**)
- Undeprecate `tail` and make alias of `rest`.

5.3.25 v1.1.0 (2014-08-19)

- Add `attempt`.
- Add `before`.
- Add `camel_case`.
- Add `capitalize`.
- Add `chunk`.
- Add `curry_right`.
- Add `drop_right`.
- Add `drop_right_while`.
- Add `drop_while`.
- Add `ends_with`.
- Add `escape_reg_exp` and `escape_re`.
- Add `is_error`.
- Add `is_reg_exp` and `is_re`.
- Add `kebab_case`.
- Add `keys_in` as alias of `keys`.
- Add `negate`.
- Add `pad`.
- Add `pad_left`.
- Add `pad_right`.

- Add `partition`.
- Add `pull_at`.
- Add `repeat`.
- Add `slice_`.
- Add `snake_case`.
- Add `sorted_last_index`.
- Add `starts_with`.
- Add `take_right`.
- Add `take_right_while`.
- Add `take_while`.
- Add `trim`.
- Add `trim_left`.
- Add `trim_right`.
- Add `trunc`.
- Add `values_in` as alias of `values`.
- Create `pydash.api.strings` module.
- Deprecate `tail`.
- Modify `drop` to accept `n` argument and remove as alias of `rest`.
- Modify `take` to accept `n` argument and remove as alias of `first`.
- Move `escape` and `unescape` from `pydash.api.utilities` to `pydash.api.strings`. (**breaking change**)
- Move `range_` from `pydash.api.arrays` to `pydash.api.utilities`. (**breaking change**)

5.3.26 v1.0.0 (2014-08-05)

- Add Python 2.6 and Python 3 support.
- Add `after`.
- Add `assign` and `extend`. Thanks nathancahill!
- Add `callback` and `create_callback`.
- Add `chain`.
- Add `clone`.
- Add `clone_deep`.
- Add `compose`.
- Add `constant`.
- Add `count_by`. Thanks nathancahill!
- Add `curry`.
- Add `debounce`.

- Add `defaults`. Thanks nathancahill!
- Add `delay`.
- Add `escape`.
- Add `find_key`. Thanks nathancahill!
- Add `find_last`. Thanks nathancahill!
- Add `find_last_index`. Thanks nathancahill!
- Add `find_last_key`. Thanks nathancahill!
- Add `for_each`. Thanks nathancahill!
- Add `for_each_right`. Thanks nathancahill!
- Add `for_in`. Thanks nathancahill!
- Add `for_in_right`. Thanks nathancahill!
- Add `for_own`. Thanks nathancahill!
- Add `for_own_right`. Thanks nathancahill!
- Add `functions_` and `methods`. Thanks nathancahill!
- Add `group_by`. Thanks nathancahill!
- Add `has`. Thanks nathancahill!
- Add `index_by`. Thanks nathancahill!
- Add `identity`.
- Add `inject`.
- Add `invert`.
- Add `invoke`. Thanks nathancahill!
- Add `is_list`. Thanks nathancahill!
- Add `is_boolean`. Thanks nathancahill!
- Add `is_empty`. Thanks nathancahill!
- Add `is_equal`.
- Add `is_function`. Thanks nathancahill!
- Add `is_none`. Thanks nathancahill!
- Add `is_number`. Thanks nathancahill!
- Add `is_object`.
- Add `is_plain_object`.
- Add `is_string`. Thanks nathancahill!
- Add `keys`.
- Add `map_values`.
- Add `matches`.
- Add `max_`. Thanks nathancahill!
- Add `memoize`.

- Add `merge`.
- Add `min_`. Thanks nathancahill!
- Add `noop`.
- Add `now`.
- Add `omit`.
- Add `once`.
- Add `pairs`.
- Add `parse_int`.
- Add `partial`.
- Add `partial_right`.
- Add `pick`.
- Add `property_` and `prop`.
- Add `pull`. Thanks nathancahill!
- Add `random`.
- Add `reduce_` and `foldl`.
- Add `reduce_right` and `foldr`.
- Add `reject`. Thanks nathancahill!
- Add `remove`.
- Add `result`.
- Add `sample`.
- Add `shuffle`.
- Add `size`.
- Add `sort_by`. Thanks nathancahill!
- Add `tap`.
- Add `throttle`.
- Add `times`.
- Add `transform`.
- Add `to_list`. Thanks nathancahill!
- Add `unescape`.
- Add `unique_id`.
- Add `values`.
- Add `wrap`.
- Add `xor`.

5.3.27 v0.0.0 (2014-07-22)

- Add `all_`.
- Add `any_`.
- Add `at`.
- Add `bisect_left`.
- Add `collect`.
- Add `collections`.
- Add `compact`.
- Add `contains`.
- Add `detect`.
- Add `difference`.
- Add `drop`.
- Add `each`.
- Add `each_right`.
- Add `every`.
- Add `filter_`.
- Add `find`.
- Add `find_index`.
- Add `find_where`.
- Add `first`.
- Add `flatten`.
- Add `head`.
- Add `include`.
- Add `index_of`.
- Add `initial`.
- Add `intersection`.
- Add `last`.
- Add `last_index_of`.
- Add `map_`.
- Add `object_`.
- Add `pluck`.
- Add `range_`.
- Add `rest`.
- Add `select`.
- Add `some`.
- Add `sorted_index`.

- Add `tail`.
- Add `take`.
- Add `union`.
- Add `uniq`.
- Add `unique`.
- Add `unzip`.
- Add `where`.
- Add `without`.
- Add `zip_`.
- Add `zip_object`.

5.4 Authors

5.4.1 Lead

- Derrick Gilland, dgilland@gmail.com, dgilland@github

5.4.2 Contributors

- Nathan Cahill, nathan@nathancahill.com, nathancahill@github
- Klaus Sevensleeper, k7sleeper@gmail.com, k7sleeper@github
- Michael James, urbnjamesmi1
- Tim Griesser, tgriesser@gmail.com, tgriesser@github

5.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/dgilland/zulu.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" or "help wanted" is open to whoever wants to implement it.

Write Documentation

Zulu could always use more documentation, whether as part of the official zulu docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dgilland/zulu.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.5.2 Get Started!

Ready to contribute? Here's how to set up `zulu` for local development.

1. Fork the `zulu` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/zulu.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenv installed, this is how you set up your fork for local development:

```
$ cd zulu
$ pip install -r requirements-dev.txt
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass linting and all unit tests by testing with tox across all supported Python versions:

```
$ invoke tox
```

6. Add yourself to `AUTHORS.rst`.
7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

5.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the README.rst.
3. The pull request should work for Python 2.7, 3.4, and 3.5. Check https://travis-ci.org/dgilland/zulu/pull_requests and make sure that the tests pass for all supported Python versions.

5.6 Kudos

Thank you to Lo-Dash for providing such a great library to port.

CHAPTER 6

Indices and Tables

- genindex
- modindex
- search

p

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z