pydash Documentation

Release 4.7.2

Derrick Gilland

Aug 08, 2018

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.map_([{'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(3, lambda index: index)
[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.map_([{'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)
```

(continues on next page)

(continued from previous page)

```
6

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

# Utilities
>>> pydash.times(3, lambda index: index)
[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 Lodash Differences

3.3.1 Naming Conventions

pydash adheres to the following conventions:

- Function names use `snake_case` instead of `camelCase`.
- Any Lodash function that shares its name with a reserved Python keyword will have an `_` appended after it (e.g. `filter` in Lodash would be `filter_` in pydash).
- Lodash's `toArray()` is pydash's `to_list()`.
- Lodash'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, to avoid import ambiguities, the `functions()` function was renamed.
- Lodash's `is_native()` is pydash's `is_builtin()`. This aligns better with Python's builtins terminology.

3.3.2 Callbacks

There are a few differences between extra callback style support:

- Pydash has an explicit shallow property access of the form `['some_property']` as in `pydash.map_([{'a.b': 1, 'a': {'b': 3}}, {'a.b': 2, 'a': {'b': 4}}], ['a.b'])` would evaulate to `[1, 2]` and not `[3, 4]` (as would be the case for `'a.b'`).

3.3.3 Extra Functions

In addition to porting Lodash, 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}
... ]
```

(continues on next page)

(continued from previous page)

```
# 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.map_(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},
```

(continues on next page)

(continued from previous page)

```
...     {'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.map_(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.get(data, 'a.b.c.2.d.1.[1]')
2

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

Pydash's callback system 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()` is called:

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

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

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

(continues on next page)

(continued from previous page)

```
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.value()
30
>>> rechained = chained.plant([5, 6, 7, 8])
>>> rechained.value()
174
>>> chained.value()
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) == pydash.add(1, 2)
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 Lodash 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 v3.x.x to v4.0.0

Start by reading the full list of changes in `v4.0.0` at the *Changelog*. There are a significant amount of backwards-incompatibilities that will likely need to be addressed:

- All function aliases have been removed in favor of having a single named function for everything. This was done to make things less confusing by having only a single named function that performs an action vs. potentially using two different names for the same function.
- A few functions have been removed whose functionality was duplicated by another function.
- Some functions have been renamed for consistency and to align with Lodash.
- Many functions have had their callback argument moved to another function to align with Lodash.
- The generic `callback` argument has been renamed to either `iteratee`, `predicate`, or `comparator`. This was done to make it clearer what the callback is doing and to align more with Lodash's naming conventions.

Once the shock of those backwards-incompatibilities has worn off, discover 72 new functions:

- 19 new array methods
  - `pydash.arrays.difference_by()`
  - `pydash.arrays.difference_with()`
  - `pydash.arrays.from_pairs()`
  - `pydash.arrays.intersection_by()`
  - `pydash.arrays.intersection_with()`
  - `pydash.arrays.nth()`
  - `pydash.arrays.pull_all()`
  - `pydash.arrays.sorted_index_by()`
  - `pydash.arrays.sorted_index_of()`
  - `pydash.arrays.sorted_last_index_by()`
  - `pydash.arrays.sorted_last_index_of()`
  - `pydash.arrays.sorted_uniq()`
  - `pydash.arrays.union_by()`
  - `pydash.arrays.union_with()`
  - `pydash.arrays.uniq_by()`
  - `pydash.arrays.uniq_with()`

    - `pydash.arrays.xor_by()`
    - `pydash.arrays.xor_with()`
    - `pydash.arrays.zip_object_deep()`
- 6 new collection methods
    - `pydash.collections.flat_map()`
    - `pydash.collections.flat_map_deep()`
    - `pydash.collections.flat_depth()`
    - `pydash.collections.flatten_depth()`
    - `pydash.collections.invoke_map()`
    - `pydash.collections.sample_size()`
- 2 new function methods
    - `pydash.functions.flip()`
    - `pydash.functions.unary()`
- 12 new object methods
    - `pydash.objects.assign_with()`
    - `pydash.objects.clone_deep_with()`
    - `pydash.objects.clone_with()`
    - `pydash.objects.invert_by()`
    - `pydash.objects.merge_with()`
    - `pydash.objects.omit_by()`
    - `pydash.objects.pick_by()`
    - `pydash.objects.set_with()`
    - `pydash.objects.to_integer()`
    - `pydash.objects.unset()`
    - `pydash.objects.update()`
    - `pydash.objects.udpate_with()`
- 8 new numerical methods
    - `pydash.numerical.clamp()`
    - `pydash.numerical.divide()`
    - `pydash.numerical.max_by()`
    - `pydash.numerical.mean_by()`
    - `pydash.numerical.min_by()`
    - `pydash.numerical.multiply()`
    - `pydash.numerical.subtract()`
    - `pydash.numerical.sum_by()`
- 4 new predicate methods

  - `pydash.predicates.eq()`
  - `pydash.predicates.is_equal_with()`
  - `pydash.predicates.is_match_with()`
  - `pydash.predicates.is_set()`
- 6 new string methods
  - `pydash.strings.lower_case()`
  - `pydash.strings.lower_first()`
  - `pydash.strings.to_lower()`
  - `pydash.strings.to_upper()`
  - `pydash.strings.upper_case()`
  - `pydash.strings.upper_first()`
- 15 new utility methods
  - `pydash.utilities.cond()`
  - `pydash.utilities.conforms()`
  - `pydash.utilities.conforms_to()`
  - `pydash.utilities.default_to()`
  - `pydash.utilities.nth_arg()`
  - `pydash.utilities.over()`
  - `pydash.utilities.over_every()`
  - `pydash.utilities.over_some()`
  - `pydash.utilities.range_right()`
  - `pydash.utilities.stub_list()`
  - `pydash.utilities.stub_dict()`
  - `pydash.utilities.stub_false()`
  - `pydash.utilities.stub_string()`
  - `pydash.utilities.stub_true()`
  - `pydash.utilities.to_path()`

3.8.2 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*.

3.8.3 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.
- 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

4.1.3 Chaining

4.1.4 Collections

4.1.5 Functions

4.1.6 Numerical

4.1.7 Objects

4.1.8 Predicates

4.1.9 Strings

4.1.10 Utilities

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 v4.7.2 (2018-08-07)

Bug Fixes

- Fix bug in `spread` where arguments were not being passed to wrapped function properly.

5.3.2 v4.7.1 (2018-08-03)

New Features

- Modify `to_dict` to first try to convert using `dict()` before falling back to using `pydash.helpers.iterator()`.

5.3.3 v4.7.0 (2018-07-26)

Misc

- Internal code optimizations.

5.3.4 v4.6.1 (2018-07-16)

Misc

- Support Python 3.7.

5.3.5 v4.6.0 (2018-07-10)

Misc

- Improve performance of the following functions for large datasets:
  - `duplicates`
  - `sorted_uniq`
  - `sorted_uniq_by`
  - `union`
  - `union_by`
  - `union_with`
  - `uniq`
  - `uniq_by`
  - `uniq_with`
  - `xor`
  - `xor_by`

  - `xor_with`

5.3.6 v4.5.0 (2018-03-20)

New Features

- Add `jitter` argument to `retry`.

5.3.7 v4.4.1 (2018-03-14)

New Features

- Add `attempt` argument to `on_exception` callback in `retry`. New function signature is `on_exception(exc, attempt)` (previously was `on_exception(exc)`). All arguments to `on_exception` callback are now optional.

5.3.8 v4.4.0 (2018-03-13)

New Features

- Add `retry` decorator that will retry a function multiple times if the function raises an exception.

5.3.9 v4.3.3 (2018-03-02)

Bug Fixes

- Fix regression in `v4.3.2` introduced by the support added for callable class callbacks that changed the handling of callbacks that could not be inspected. Prior to `v4.3.2`, these callbacks would default to being passed a single callback argument, but with `v4.3.2` these callbacks would be passed the full set of callback arguments which could result an exception being raised due to the callback not supporting that many arguments.

5.3.10 v4.3.2 (2018-02-06)

Bug Fixes

- Fix issue in `defaults_deep` where sources with non-dict values would raise an exception due to assumption that object was always a dict.
- Fix issue in `curry` where too many arguments would be passed to the curried function when evaluating function if too many arguments used in last function call.
- Workaround issue in Python 2.7 where callable classes used as callbacks were always passed the full count of arguments even when the callable class only accept a subset of arguments.

5.3.11 v4.3.1 (2017-12-19)

Bug Fixes

- Fix set_with so that callable values are not called when being set. This bug also impacted the following functions by proxy:
  - pick
  - pick_by
  - set_
  - transpose
  - zip_object_deep

5.3.12 v4.3.0 (2017-11-22)

New Features

- Add nest.
- Wrap non-iterables in a list in to_list instead of raising an exception. Thanks efenka!
- Add split_strings argument to to_list to control whether strings are coverted to a list (split_strings=True) or wrapped in a list (split_strings=False). Default is split_strings=True. Thanks efenka!

5.3.13 v4.2.1 (2017-09-08)

Bug Fixes

- Ensure that to_path always returns a list.
- Fix get to work with path values other than just strings, integers, and lists.

5.3.14 v4.2.0 (2017-09-08)

New Features

- Support more iterator "hooks" in to_dict so non-iterators that expose an items(), iteritems(), or has __dict__ attributes will be converted using those methods.
- Support deep paths in omit and omit_by. Thanks beck3905!
- Support deep paths in pick and pick_by. Thanks beck3905!

Bug Fixes

- Fix missing argument passing to matched function in cond.
- Support passing a single list of pairs in cond instead of just pairs as separate arguments.

5.3.15 v4.1.0 (2017-06-09)

New Features

- Officially support Python 3.6.
- Add `properties` function that returns list of path values for an object.
- Add `replace_end`.
- Add `replace_start`.
- Make `iteratee` support `properties`-style callback when a `tuple` is passed.
- Make `replace` accept `from_start` and `from_end` arguments to limit replacement to start and/or end of string.

Bug Fixes

- None

5.3.16 v4.0.4 (2017-05-31)

New Features

- None

Bug Fixes

- Improve performance of `get`. Thanks shaunpatterson!

5.3.17 v4.0.3 (2017-04-20)

New Features

- None

Bug Fixes

- Fix regression in `get` where `list` and `dict` objects had attributes returned when a key was missing but the key corresponded to an attribute name. For example, `pydash.get({}, 'update')` would return `{}.update()` instead of `None`. Previous behavior was that only item-access was allowed for `list` and `dict` which has been restored.
- Fix regression in `invoke`/`invoke_map` where non-attributes could be invoked. For example, `pydash.invoke({'items': lambda: 1}, 'items')` would return `1` instead of `dict_items([('a', 'items')])`. Previous behavior was that only attribute methods could be invoked which has now been restored.

5.3.18 v4.0.2 (2017-04-04)

New Features

- None

Bug Fixes

- Fix regression in `intersection`, `intersection_by`, and `intersection_with` introduced in `v4.0.0` where the a single argument supplied to intersection should return the same argument value instead of an empty list.

Backwards-Incompatibilities

- None

5.3.19 v4.0.1 (2017-04-04)

New Features

- Make `property_` work with deep path strings.

Bug Fixes

- Revert removal of `deep_pluck` and rename to `pluck`. Previously, `deep_pluck` was removed and `map_` was recommended as a replacement. However, `deep_pluck` (now defined as `pluck`) functionality is not supported by `map_` so the removal `pluck` was reverted.

Backwards-Incompatibilities

- Remove `property_deep` (use `property_`).

5.3.20 v4.0.0 (2017-04-03)

New Features

- Add `assign_with`.
- Add `clamp`.
- Add `clone_deep_with`.
- Add `clone_with`.
- Add `cond`. Thanks bharadwajyarlagadda!
- Add `conforms`.
- Add `conforms_to`.
- Add `default_to`. Thanks bharadwajyarlagadda!
- Add `difference_by`.

- Add `difference_with`.
- Add `divide`. Thanks bharadwajyarlagadda!
- Add `eq`. Thanks bharadwajyarlagadda!
- Add `flat_map`.
- Add `flat_map_deep`.
- Add `flat_map_depth`.
- Add `flatten_depth`.
- Add `flip`. Thanks bharadwajyarlagadda!
- Add `from_pairs`. Thanks bharadwajyarlagadda!
- Add `intersection_by`.
- Add `intersection_with`.
- Add `invert_by`.
- Add `invoke_map`.
- Add `is_equal_with`. Thanks bharadwajyarlagadda!
- Add `is_match_with`.
- Add `is_set`. Thanks bharadwajyarlagadda!
- Add `lower_case`. Thanks bharadwajyarlagadda!
- Add `lower_first`. Thanks bharadwajyarlagadda!
- Add `max_by`.
- Add `mean_by`.
- Add `merge_with`.
- Add `min_by`.
- Add `multiply`. Thanks bharadwajyarlagadda!
- Add `nth`. Thanks bharadwajyarlagadda!
- Add `nth_arg`. Thanks bharadwajyarlagadda!
- Add `omit_by`.
- Add `over`. Thanks bharadwajyarlagadda!
- Add `over_every`. Thanks bharadwajyarlagadda!
- Add `over_some`. Thanks bharadwajyarlagadda!
- Add `pick_by`.
- Add `pull_all`. Thanks bharadwajyarlagadda!
- Add `pull_all_by`.
- Add `pull_all_with`.
- Add `range_right`. Thanks bharadwajyarlagadda!
- Add `sample_size`. Thanks bharadwajyarlagadda!
- Add `set_with`.

- Add `sorted_index_by`.
- Add `sorted_index_of`. Thanks bharadwajyarlagadda!
- Add `sorted_last_index_by`.
- Add `sorted_last_index_of`.
- Add `sorted_uniq`. Thanks bharadwajyarlagadda!
- Add `sorted_uniq_by`.
- Add `stub_list`. Thanks bharadwajyarlagadda!
- Add `stub_dict`. Thanks bharadwajyarlagadda!
- Add `stub_false`. Thanks bharadwajyarlagadda!
- Add `stub_string`. Thanks bharadwajyarlagadda!
- Add `stub_true`. Thanks bharadwajyarlagadda!
- Add `subtract`. Thanks bharadwajyarlagadda!
- Add `sum_by`.
- Add `to_integer`.
- Add `to_lower`. Thanks bharadwajyarlagadda!
- Add `to_path`. Thanks bharadwajyarlagadda!
- Add `to_upper`. Thanks bharadwajyarlagadda!
- Add `unary`.
- Add `union_by`. Thanks bharadwajyarlagadda!
- Add `union_with`. Thanks bharadwajyarlagadda!
- Add `uniq_by`.
- Add `uniq_with`.
- Add `unset`.
- Add `update`.
- Add `update_with`.
- Add `upper_case`. Thanks bharadwajyarlagadda!
- Add `upper_first`. Thanks bharadwajyarlagadda!
- Add `xor_by`.
- Add `xor_with`.
- Add `zip_object_deep`.
- Make function returned by `constant` ignore extra arguments when called.
- Make `get` support attribute access within path.
- Make `iteratee` treat an integer argument as a string path (i.e. `iteratee(1)` is equivalent to `iteratee('1')` for creating a path accessor function).
- Make `intersection` work with unhashable types.
- Make `range_` support decrementing when `start` argument is greater than `stop` argument.

- Make `xor` maintain sort order of supplied arguments.

Bug Fixes

- Fix `find_last_key` so that it iterates over object in reverse.

Backwards-Incompatibilities

- Make `add` only support two argument addition. (**breaking change**)
- Make `difference` return duplicate values from first argument and maintain sort order. (**breaking change**)
- Make `invoke` work on objects instead of collections. Use `invoke_map` for collections. (**breaking change**)
- Make `set_` support mixed `list/dict` defaults within a single object based on whether key or index path substrings used. (**breaking change**)
- Make `set_` modify object in place. (**breaking change**)
- Only use `merge` callback result if result is not `None`. Previously, result from callback (if provided) was used unconditionally. (**breaking change**)
- Remove functions: (**breaking change**)
  - `deep_pluck` (no alternative) [**UPDATE:** `deep_pluck` functionality restored as `pluck` in `v4.0.1`]
  - `mapiter` (no alternative)
  - `pluck` (use `map_`)
  - `update_path` (use `update` or `update_with`)
  - `set_path` (use `set_` or `set_with`)
- Remove aliases: (**breaking change**)
  - `all_` (use `every`)
  - `any_` (use `some`)
  - `append` (use `push`)
  - `average` and `avg` (use `mean` or `mean_by`)
  - `callback` (use `iteratee`)
  - `cat` (use `concat`)
  - `collect` (use `map_`)
  - `contains` (use `includes`)
  - `curve` (use `round_`)
  - `deep_get` and `get_path` (use `get`)
  - `deep_has` and `has_path` (use `has`)
  - `deep_prop` (use `property_deep`)
  - `deep_set` (use `set_`)
  - `detect` and `find_where` (use `find`)
  - `each` (use `for_each`)

- each_right (use for_each_right)
- escape_re (use escape_reg_exp)
- explode (use split)
- extend (use assign)
- first (use head)
- foldl (use reduce)
- foldr (use reduce_right)
- for_own (use for_each)
- for_own_right (use for_each_right)
- implode (use join)
- is_bool (use is_boolean)
- is_int (use is_integer)
- is_native (use is_builtin)
- is_num (use is_number)
- is_plain_object (use is_dict)
- is_re (use is_reg_exp)
- js_match (use reg_exp_js_match)
- js_replace (use reg_exp_js_replace)
- keys_in (use keys)
- moving_average and moving_avg (use moving_mean)
- object_ (use zip_object)
- pad_left (use pad_start)
- pad_right (use pad_end)
- pipe (use flow)
- pipe_right and compose (use flow_right)
- prop (use property_)
- prop_of (use property_of)
- pow_ (use power)
- re_replace (use reg_exp_replace)
- rest (use tail)
- select (use filter_)
- sigma (use std_deviation)
- sort_by_all and sort_by_order (use order_by)
- trim_left (use trim_start)
- trim_right (use trim_right)
- trunc (use truncate)

  - `underscore_case` (use `snake_case`)
  - `unique` (use `uniq`)
  - `values_in` (use `values`)
  - `where` (use `filter_`)
- Rename functions: (**breaking change**)
  - `deep_map_values` to `map_values_deep`
  - `deep_property` to `property_deep`
  - `include` to `includes`
  - `index_by` to `key_by`
  - `mod_args` to `over_args`
  - `moving_average` to `moving_mean`
  - `pairs` to `to_pairs`
- Remove `callback` argument from: (**breaking change**)
  - `assign`. Moved to `assign_with`.
  - `clone` and `clone_deep`. Moved to `clone_with` and `clone_deep_with`.
  - `is_match`. Moved to `is_match_with`.
  - `max_` and `min_`. Moved to `max_by` and `min_by`.
  - `omit`. Moved to `omit_by`.
  - `pick`. Moved to `pick_by`.
  - `sorted_index`. Moved to `sorted_index_by`.
  - `sum_`. Moved to `sum_by`.
  - `uniq`/`unique`. Moved to `uniq_by`.
- Renamed `callback` argument to `predicate`: (**breaking change**)
  - `drop_right_while`
  - `drop_while`
  - `every`
  - `filter_`
  - `find`
  - `find_key`
  - `find_last`
  - `find_index`
  - `find_last_index`
  - `find_last_key`
  - `partition`
  - `reject`
  - `remove`

  - some
  - take_right_while
  - take_while
- Renamed `callback` argument to `iteratee`: (**breaking change**)
  - count_by
  - duplicates
  - for_each
  - for_each_right
  - for_in
  - for_in_right
  - group_by
  - key_by
  - map_
  - map_keys
  - map_values
  - map_values_deep
  - mapcat
  - median
  - reduce_
  - reduce_right
  - reductions
  - reductions_right
  - sort_by
  - times
  - transform
  - unzip_with
  - zip_with
  - zscore
- Rename `comparison` argument in `sort` to `comparator`.
- Rename `index` and `how_many` arguments in `splice` to `start` and `count`.
- Remove `multivalue` argument from `invert`. Feature moved to `invert_by`. (**breaking change**)

5.3.21 v3.4.8 (2017-01-05)

- Make internal function inspection methods work with Python 3 annotations. Thanks tgriesser!

5.3.22 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!

5.3.23 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!

5.3.24 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`.

5.3.25 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`.

5.3.26 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.

5.3.27 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.

5.3.28 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`.

5.3.29 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.

5.3.30 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.

5.3.31 v3.2.2 (2015-04-29)

- Catch `AttributeError` in `helpers.get_item` and return default value if set.

5.3.32 v3.2.1 (2015-04-29)

- Fix bug in `reduce_right` where collection was not reversed correctly.

5.3.33 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.

5.3.34 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,)`.

5.3.35 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.

5.3.36 v2.4.2 (2015-02-03)

- Fix `remove` so that array is modified after callback iteration.

5.3.37 v2.4.1 (2015-01-11)

- Fix `kebab_case` so that it casts string to lower case.

5.3.38 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!

5.3.39 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.

5.3.40 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 `_`.

5.3.41 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.

5.3.42 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`.

5.3.43 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`.

5.3.44 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.

5.3.45 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**)

5.3.46 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`.

5.3.47 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
- Bharadwaj Yarlagadda, yarlagaddabharadwaj@gmail.com, bharadwajyarlagadda@github
- Michael James, urbnjamesmi1@github
- Tim Griesser, tgriesser@gmail.com, tgriesser@github
- Shaun Patterson, shaunpatterson@github
- Brian Beck, beck3905@github
- Frank Epperlein, efenka@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/pydash.

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

pydash could always use more documentation, whether as part of the official pydash 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/pydash.

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 `pydash` for local development.

1. Fork the `pydash` repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_username_here/pydash.git
   ```

3. Install Python dependencies into a virtualenv:

   ```
   $ cd pydash
   $ 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:

```
$ 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 all versions Python that this project supports. Check https://travis-ci.org/dgilland/pydash/pull_requests and make sure that the all environments pass.

5.6 Kudos

Thank you to Lodash for providing such a great library to port.

CHAPTER 6

Indices and Tables

- genindex
- modindex
- search