

---

# **yail Documentation**

***Release 0.0.2+0.ga2dc1de.dirty***

**John Kirkham**

**Nov 12, 2017**



---

## Contents

---

<b>1</b>	<b>yail</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>5</b>
<b>3</b>	<b>Usage</b>	<b>7</b>
<b>4</b>	<b>API</b>	<b>9</b>
<b>5</b>	<b>Contributing</b>	<b>15</b>
<b>6</b>	<b>Indices and tables</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>



Contents:



Yet Another Iterator Library for Python.

- Free software: BSD 3-Clause
- Documentation: <https://yail.readthedocs.io>.

## 1.1 Features

- TODO

## 1.2 Credits

This package was created with [Cookiecutter](#) and the [nanshe-org/nanshe-cookiecutter](#) project template.





### 2.1 Stable release

To install `yail`, run this command in your terminal:

```
$ pip install yail
```

This is the preferred method to install `yail`, as it will always install the most recent stable release.

If you don't have `pip` installed, this [Python installation guide](#) can guide you through the process.

### 2.2 From sources

The sources for `yail` can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/jakirkham/yail
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/jakirkham/yail/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```



## CHAPTER 3

---

### Usage

---

To use yail in a project:

```
import yail
```



## 4.1 `yail` package

### 4.1.1 Submodules

#### `yail.core` module

`yail.core.cycles(seq, n=1)`

Cycles through the sequence *n*-times.

Basically the same as `itertools.cycle` except that this sets an upper limit on how many cycles will be done.

---

**Note:** If *n* is *None*, this is identical to `itertools.cycle`.

---

#### Parameters

- **seq** (*iterable*) – The sequence to grab items from.
- **n** (*integral*) – Number of times to cycle through.

**Returns** The cycled sequence generator.

**Return type** *generator*

#### Examples

```
>>> list(cycles([1, 2, 3], 2))
[1, 2, 3, 1, 2, 3]
```

`yail.core.disperse(seq)`

Similar to `range` except that it recursively proceeds through the given range in such a way that values that follow each other are preferably not only non-sequential, but fairly different. This does not always work with small ranges, but works nicely with large ranges.

**Parameters**

- **a** (*int*) – the lower bound of the range
- **b** (*int*) – the upper bound of the range

**Returns**

a generator that can be used to iterate through the sequence.

**Return type** `result(generator)`

## Examples

```
>>> list(disperse(range(10)))
[0, 5, 8, 3, 9, 4, 6, 1, 7, 2]
```

`yail.core.duplicate(seq, n=1)`

Gets each element multiple times.

Like `itertools.repeat` this will repeat each element n-times. However, it will do this for each element of the sequence.

**Parameters**

- **seq** (*iterable*) – The sequence to grab items from.
- **n** (*integral*) – Number of repeats for each element.

**Returns** A generator of repeated elements.

**Return type** `generator`

## Examples

```
>>> list(duplicate([1, 2, 3], 2))
[1, 1, 2, 2, 3, 3]
```

`yail.core.empty()`

Creates an empty iterator.

## Examples

```
>>> list(empty())
[]
```

`yail.core.generator(it)`

Creates a generator type from the iterable.

**Parameters** **it** (*iterable*) – An iterable to make a generator.

**Returns** A generator made from the iterable.

**Return type** *generator*

### Examples

```
>>> generator(range(5))
<generator object generator at 0x...>
```

```
>>> list(generator(range(5)))
[0, 1, 2, 3, 4]
```

`yail.core.indices(*sizes)`

Iterates over a length/shape.

Takes a size or sizes (unpacked shape) and iterates through all combinations of the indices.

**Parameters** *\*sizes* (*int*) – list of sizes to iterate over.

**Returns** an iterator over the sizes.

**Return type** iterable

### Examples

```
>>> list(indices(3, 2))
[(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1)]
```

`yail.core.pad(seq, before=0, after=0, fill=None)`

Pads a sequence by a fill value before and/or after.

Pads the sequence before and after using the fill value provided by `fill` up to the lengths specified by `before` and `after`. If either `before` or `after` is `None`, pad the fill value infinitely on the respective end.

---

**Note:** If `before` is `None`, the sequence will only be the fill value.

---

#### Parameters

- **seq** (*iterable*) – Sequence to pad.
- **before** (*integral*) – Amount to pad before.
- **after** (*integral*) – Amount to pad after.
- **fill** (*any*) – Some value to pad with.

**Returns** A sequence that has been padded.

**Return type** iterable

### Examples

```
>>> list(pad(range(2, 4), before=1, after=2, fill=0))
[0, 2, 3, 0, 0]
```

`yail.core.single(val)`

Creates an iterator with a single value.

**Parameters** `val` (*any*) – Single value to add to the iterator.

**Returns** An iterable yielding the single value.

**Return type** iterable

## Examples

```
>>> list(single(1))
[1]
```

`yail.core.sliding_window_filled(seq, n, pad_before=False, pad_after=False, fillvalue=None)`

A sliding window with optional padding on either end..

### Parameters

- `seq` (*iter*) – an iterator or something that can be turned into an iterator
- `n` (*int*) – number of generators to create as lagged
- `pad_before` (*bool*) – whether to continue zipping along the longest generator
- `pad_after` (*bool*) – whether to continue zipping along the longest generator
- `fillvalue` – value to use to fill generators shorter than the longest.

### Returns

a generator object that will return values from each iterator.

**Return type** generator object

## Examples

```
>>> list(sliding_window_filled(range(5), 2))
[(0, 1), (1, 2), (2, 3), (3, 4)]
```

```
>>> list(sliding_window_filled(range(5), 2, pad_after=True))
[(0, 1), (1, 2), (2, 3), (3, 4), (4, None)]
```

```
>>> list(sliding_window_filled(range(5), 2, pad_before=True, pad_after=True))
[(None, 0), (0, 1), (1, 2), (2, 3), (3, 4), (4, None)]
```

`yail.core.split(n, seq)`

Splits the sequence around element `n`.

Provides 3 “iterable”s in return.

1. Everything before the `n`-th value.
2. An iterable with just the `n`-th value.
3. Everything after the `n`-th value.

### Parameters

- `n` (*integral*) – Index to split the iterable at.
- `seq` (*iterable*) – The sequence to split.



**Returns**

Each portion of the iterable around the index.

**Return type** tuple of “iterable”s

**Examples**

```
>>> list(map(tuple, split(2, range(5))))
[(0, 1), (2,), (3, 4)]
```

```
>>> list(map(tuple, split(2, [10, 20, 30, 40, 50])))
[(10, 20), (30,), (40, 50)]
```

`yail.core.subrange` (*start*, *stop=None*, *step=None*, *substep=None*)

Generates start and stop values for each subrange.

**Parameters**

- **start** (*int*) – First value in range (or last if only specified value)
- **stop** (*int*) – Last value in range
- **step** (*int*) – Step between each range
- **substep** (*int*) – Step within each range

**Yields** *range* – A subrange within the larger range.

**Examples**

```
>>> list(map(list, subrange(5)))
[[0], [1], [2], [3], [4]]
```

```
>>> list(map(list, subrange(0, 12, 3, 2)))
[[0, 2], [3, 5], [6, 8], [9, 11]]
```



Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at <https://github.com/jakirkham/yail/issues>.

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.

### 5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

### 5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

### 5.1.4 Write Documentation

yail could always use more documentation, whether as part of the official yail docs, in docstrings, or even on the web in blog posts, articles, and such.

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/jakirkham/yail/issues>.

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 :)

## 5.2 Get Started!

Ready to contribute? Here's how to set up *yail* for local development.

1. Fork the *yail* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/yail.git
```

3. Install your local copy into an environment. Assuming you have conda installed, this is how you set up your fork for local development (on Windows drop *source*). Replace “<some version>” with the Python version used for testing.:

```
$ conda create -n yailenv python="<some version>"
$ source activate yailenv
$ python setup.py develop
```

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 flake8 and the tests, including testing other Python versions:

```
$ flake8 yail tests
$ python setup.py test or py.test
```

To get flake8, just conda install it into your environment.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 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 list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5, and 3.6. Check [https://travis-ci.org/jakirkham/yail/pull\\_requests](https://travis-ci.org/jakirkham/yail/pull_requests) and make sure that the tests pass for all supported Python versions.

## 5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_yail
```



## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





**y**

yail, 9

yail.core, 9



### C

`cycles()` (in module `yail.core`), 9

### D

`disperse()` (in module `yail.core`), 9

`duplicate()` (in module `yail.core`), 10

### E

`empty()` (in module `yail.core`), 10

### G

`generator()` (in module `yail.core`), 10

### I

`indices()` (in module `yail.core`), 11

### P

`pad()` (in module `yail.core`), 11

### S

`single()` (in module `yail.core`), 11

`sliding_window_filled()` (in module `yail.core`), 12

`split()` (in module `yail.core`), 12

`subrange()` (in module `yail.core`), 13

### Y

`yail` (module), 9

`yail.core` (module), 9