# Coconut Contributing Guidelines

\_By contributing to Coconut, you agree to your contribution being released under [Coconut's Apache 2.0 license](https://github.com/evhub/coconut/blob/develop/LICENSE.txt).\_

\*\*Anyone is welcome to submit an issue or pull request!\*\* The purpose of this document is simply to explain the contribution process and the internals of how Coconut works to make contributing easier.

If you are considering contributing to Coconut, you'll be doing so on the [`develop` branch](https://github.com/evhub/coconut/tree/develop), which means you should be viewing [the `develop` version of the Contributing Guidelines](http://coconut.readthedocs.io/en/develop/CONTRIBUTING.html), if you aren't doing so already.

## Asking Questions

If you are thinking about contributing to Coconut, please don't hesitate to ask questions at Coconut's [Gitter](https://gitter.im/evhub/coconut)! That includes any questions at all about contributing, including understanding the source code, figuring out how to implement a specific change, or just trying to figure out what needs to be done.

## Bounties

Coconut development is monetarily supported by Coconut's [Backers](https://opencollective.com/coconut#backer) and [Sponsors](https://opencollective.com/coconut#sponsor) on Open Collective. As a result of this, many of Coconut's open issues are [labeled](https://github.com/evhub/coconut/labels) with bounties denoting the compensation available for resolving them. If you successfully resolve one of these issues (defined as getting a pull request resolving the issue merged), you become eligible to collect that issue's bounty. To do so, simply [file an expense report](https://opencollective.com/coconut/expenses/new#) for the correct amount with a link to the issue you resolved.

If an issue you really want fixed or an issue you're really excited to work on doesn't currently have a bounty on it, please leave a comment on the issue! Bounties are flexible, and some issues will always fall through the cracks, so don't be afraid to just ask if an issue doesn't have a bounty and you want it to.

## Good First Issues

Want to help out, but don't know what to work on? Head over to Coconut's [open issues](https://github.com/evhub/coconut/issues) and look for ones labeled "good first issue." These issues are those that require less intimate knowledge of Coconut's inner workings, and are thus possible for new contributors to work on.

## Contribution Process

Contributing to Coconut is as simple as

1. forking Coconut on [GitHub](https://github.com/evhub/coconut),

2. making changes to the [`develop` branch](https://github.com/evhub/coconut/tree/develop), and

3. proposing a pull request.

\_Note: Don't forget to add yourself to the "Authors:" section in the moduledocs of any files you modify!\_

## Testing New Changes

First, you'll want to set up a local copy of Coconut's recommended development environment. For that, just run `git checkout develop` and `make dev`. That should switch you to the `develop` branch, install all possible dependencies, bind the `coconut` command to your local copy, and set up [pre-commit](http://pre-commit.com/), which will check your code for errors for you whenever you `git commit`.

Then, you should be able to use the Coconut command-line for trying out simple things, and to run a paired-down version of the test suite locally, just `make test-basic`.

After you've tested your changes locally, you'll want to add more permanent tests to Coconut's test suite. Coconut's test suite is primarily written in Coconut itself, so testing new features just means using them inside of one of Coconut's `.coco` test files, with some `assert` statements to check validity.

## File Layout

- `DOCS.md`

+ Markdown file containing detailed documentation on every Coconut feature. If you are adding a new feature, you should also add documentation on it to this file.

- `FAQ.md`

+ Markdown file containing frequently asked questions and their answers. If you had a question you wished was answered earlier when learning Coconut, you should add it to this file.

- `HELP.md`

+ Markdown file containing Coconut's tutorial. The tutorial should be a streamlined introduction to Coconut and all of its most important features.

- `Makefile`

+ Contains targets for installing Coconut, building the documentation, checking for dependency updates, etc.

- `setup.py`

+ Using information from `requirements.py` and `constants.py` to install Coconut. Also reads `README.rst` to generate the PyPI description.

- `conf.py`

+ Sphinx configuration file for Coconut's documentation.

- coconut

- `\_\_coconut\_\_.py`

+ Mimics the Coconut header by generating and executing it when imported. Used by the REPL.

- `\_\_init\_\_.py`

+ Includes the implementation of the `%coconut` IPython magic.

- `\_\_main\_\_.py`

+ Imports and runs `main` from `main.py`.

- `constants.py`

+ All constants used across Coconut are defined here, including dependencies, magic numbers/strings, etc.

- `convenience.py`

+ Contains `cmd`, `version`, `setup`, and `parse` functions as convenience utilities when using Coconut as a module. Documented in `DOCS.md`.

- `exceptions.py`

+ All of the exceptions raised by Coconut are defined here, both those shown to the user and those used only internally.

- `highlighter.py`

+ Contains Coconut's Pygments syntax highlighter, as well as modified Python highlighters that don't fail if they encounter unknown syntax.

- `main.py`

+ Contains `main` and `main\_run`, the entry points for the `coconut` and `coconut-run` commands, respectively.

- `requirements.py`

+ Processes Coconut's requirements from `constants.py` into a form `setup.py` can use, as well as checks for updates to Coconut's dependencies.

- `root.py`

+ `root.py` creates and executes the part of Coconut's header that normalizes Python built-ins across versions. Whenever you are writing a new file, you should always add `from coconut.root import \*` to ensure compatibility with different Python versions. `root.py` also sets basic version-related constants.

- `terminal.py`

+ Contains utilities for displaying messages to the console, mainly `logger`, which is Coconut's primary method of logging a message from anywhere.

- command

- `\_\_init\_\_.py`

+ Imports everything in `command.py`.

- `cli.py`

+ Creates the `ArgumentParser` object used to parse Coconut command-line arguments.

- `command.py`

+ Contains `Command`, whose `start` method is the main entry point for the Coconut command-line utility.

- `mypy.py`

+ Contains objects necessary for Coconut's `--mypy` flag.

- `util.py`

+ Contains utilities used by `command.py`, including `Prompt` for getting syntax-highlighted input, and `Runner` for executing compiled Python.

- `watch.py`

+ Contains objects necessary for Coconut's `--watch` flag.

- compiler

- `\_\_init\_\_.py`

+ Imports everything in `compiler.py`.

- `compiler.py`

+ Contains `Compiler`, the class that actually compiles Coconut code. `Compiler` inherits from `Grammar` in `grammar.py` to get all of the basic grammatical definitions, then extends them with all of the handlers that depend on the compiler's options (e.g. the current `--target`). `Compiler` also does pre- and post-processing, including replacing strings with markers (pre-processing) and adding the header (post-processing).

- `grammar.py`

+ Contains `Grammar`, the class that specifies Coconut's grammar in PyParsing. Coconut performs one-pass compilation by attaching "handlers" to specific grammar objects to transform them into compiled Python. `grammar.py` contains all basic (non-option-dependent) handlers.

- `header.py`

+ Contains `getheader`, which generates the header at the top of all compiled Coconut files.

- `matching.py`

+ Contains `Matcher`, which handles the compilation of all Coconut pattern-matching, including `match` statements, destructuring assignment, and pattern-matching functions.

- `util.py`

+ Contains utilities for working with PyParsing objects that are primarily used by `grammar.py`.

- templates

- `header.py\_template`

+ Template for the main body of Coconut's header; use and formatting of this file is all in `header.py`.

- icoconut

- `\_\_init\_\_.py`

+ Imports everything from `icoconut/root.py`.

- `\_\_main\_\_.py`

+ Contains the main entry point for Coconut's Jupyter kernel.

- `root.py`

+ Contains the implementation of Coconut's Jupyter kernel, made by subclassing the IPython kernel.

- stubs

- `\_\_coconut\_\_.pyi`

+ A MyPy stub file for specifying the type of all the objects defined in Coconut's package header (which is saved as `\_\_coconut\_\_.py`).

- tests

- `\_\_init\_\_.py`

+ Imports everything in `main\_test.py`.

- `\_\_main\_\_.py`

+ When run, compiles all of the test source code, but \_does not run any tests\_. To run the tests, the command `make test`, or a `pytest` command to run a specific test, is necessary.

- `main\_test.py`

+ Contains `TestCase` subclasses that run all of the commands for testing the Coconut files in `src`.

- src

- `extras.coco`

+ Directly imports and calls functions in the Coconut package, including from `convenience.py` and icoconut.

- `runnable.coco`

+ Makes sure the argument `--arg` was passed when running the file.

- `runner.coco`

+ Runs `main` from `cocotest/agnostic/main.py`.

- cocotest

+ \_Note: Files in the folders below all get compiled into the top-level cocotest directory. The folders are only for differentiating what files to compile on what Python version.\_

- agnostic

- `\_\_init\_\_.coco`

+ Contains a docstring that `main.coco` asserts exists.

- `main.coco`

+ Contains the main test entry point as well as many simple, one-line tests.

- `specific.coco`

+ Tests to be run only on a specific Python version, but not necessarily only under a specific `--target`.

- `suite.coco`

+ Tests objects defined in `util.coco`.

- `tutorial.coco`

+ Tests all the examples in `TUTORIAL.md`.

- `util.coco`

+ Contains objects used in `suite.coco`.

- python2

- `py2\_test.coco`

+ Tests to be run only on Python 2 with `--target 2`.

- python3

- `py3\_test.coco`

+ Tests to be run only on Python 3 with `--target 3`.

- python35

- `py35\_test.coco`

+ Tests to be run only on Python 3.5 with `--target 3.5`.

- python36

- `py36\_test.coco`

+ Tests to be run only on Python 3.6 with `--target 3.6`.

## Release Process

1. Preparation:

1. Run `make check-reqs` and update dependencies as necessary

1. Run `make format`

1. Check changes in [`compiled-cocotest`](https://github.com/evhub/compiled-cocotest), [`pyprover`](https://github.com/evhub/pyprover), and [`coconut-prelude`](https://github.com/evhub/coconut-prelude)

1. Check [Codebeat](https://codebeat.co/a/evhub/projects) and [LGTM](https://lgtm.com/dashboard) for `coconut` and `compiled-cocotest`

1. Make sure [`coconut-develop`](https://pypi.python.org/pypi/coconut-develop) package looks good

1. Run `make docs` and ensure local documentation looks good

1. Make sure [develop documentation](http://coconut.readthedocs.io/en/develop/) looks good

1. Make sure [Travis](https://travis-ci.org/evhub/coconut/builds) and [AppVeyor](https://ci.appveyor.com/project/evhub/coconut) are passing

1. Run `make test-easter-eggs`

1. Turn off `develop` in `root.py`

1. Set `root.py` to new version number

1. If major release, set `root.py` to new version name

2. Pull Request:

1. Create a pull request to merge `develop` into `master`

1. Link contributors on pull request

1. Wait until everything is passing

3. Release:

1. Release [`sublime-coconut`](https://github.com/evhub/sublime-coconut) first if applicable

1. Merge pull request and mark as resolved

1. Release `master` on GitHub

1. `git fetch`, `git checkout master`, and `git pull`

1. Run `make upload`

1. `git checkout develop`, `git rebase master`, and `git push`

1. Turn on `develop` in `root`

1. Run `make dev`

1. Push to `develop`

1. Wipe all updated versions on [readthedocs](https://readthedocs.org/projects/coconut/versions/)

1. Build all updated versions on [readthedocs](https://readthedocs.org/projects/coconut/builds/)

1. Copy [PyPI](https://pypi.python.org/pypi/coconut) keywords to [readthedocs tags](https://readthedocs.org/dashboard/coconut/edit/)

1. Get SHA-256 hash from [PyPI](https://pypi.python.org/pypi/coconut) `.tar.gz` file and use that as well as the current version requirements in [`constants.py`](https://github.com/evhub/coconut/blob/master/coconut/constants.py) to update the [local feedstock](https://github.com/evhub/coconut-feedstock)

1. Submit PR to update [Coconut's `conda-forge` feedstock](https://github.com/conda-forge/coconut-feedstock)

1. Update [website](https://github.com/evhub/coconut/tree/gh-pages) if it needs updating

1. Wait until feedstock PR is passing then merge it

1. Close release [milestone](https://github.com/evhub/coconut/milestones?direction=asc&sort=due\_date)