# Contributor's Guide

If you're reading this, you're probably looking to contributing to Camelot. \*Time is the only real currency\*, and the fact that you're considering spending some here is \*very\* generous of you. Thank you very much!

This document will help you get started with contributing documentation, code, testing and filing issues. If you have any questions, feel free to reach out to [Vinayak Mehta](https://vinayak-mehta.github.io), the author and maintainer.

## Code Of Conduct

The following quote sums up the \*\*Code Of Conduct\*\*.

> Be cordial or be on your way. --Kenneth Reitz

Kenneth Reitz has also written an [essay](https://www.kennethreitz.org/essays/be-cordial-or-be-on-your-way) on this topic, which you should read.

As the [Requests Code Of Conduct](http://docs.python-requests.org/en/master/dev/contributing/#be-cordial) states, \*\*all contributions are welcome\*\*, as long as everyone involved is treated with respect.

## Your first contribution

A great way to start contributing to Camelot is to pick an issue tagged with the [help wanted](https://github.com/camelot-dev/camelot/labels/help%20wanted) tag or the [good first issue](https://github.com/camelot-dev/camelot/labels/good%20first%20issue) tag. If you're unable to find a good first issue, feel free to contact the maintainer.

## Setting up a development environment

To install the dependencies needed for development, you can use pip:

<pre>

$ pip install camelot-py[dev]

</pre>

Alternatively, you can clone the project repository, and install using pip:

<pre>

$ pip install ".[dev]"

</pre>

## Pull Requests

### Submit a pull request

The preferred workflow for contributing to Camelot is to fork the [project repository](https://github.com/camelot-dev/camelot) on GitHub, clone, develop on a branch and then finally submit a pull request. Here are the steps:

1. Fork the project repository. Click on the ‘Fork’ button near the top of the page. This creates a copy of the code under your account on the GitHub.

2. Clone your fork of Camelot from your GitHub account:

<pre>

$ git clone https://www.github.com/[username]/camelot

</pre>

3. Create a branch to hold your changes:

<pre>

$ git checkout -b my-feature

</pre>

Always branch out from `master` to work on your contribution. It's good practice to never work on the `master` branch!

\*\*Protip: `git stash` is a great way to save the work that you haven't committed yet, to move between branches.\*\*

4. Work on your contribution. Add changed files using `git add` and then `git commit` them:

<pre>

$ git add modified\_files

$ git commit

</pre>

5. Finally, push them to your GitHub fork:

<pre>

$ git push -u origin my-feature

</pre>

Now it's time to go to the your fork of Camelot and create a pull request! You can [follow these instructions](https://help.github.com/articles/creating-a-pull-request-from-a-fork/) to do this.

### Work on your pull request

We recommend that your pull request complies with the following rules:

- Make sure your code follows [pep8](http://pep8.org).

- In case your pull request contains function docstrings, make sure you follow the [numpydoc](https://numpydoc.readthedocs.io/en/latest/format.html) format. All function docstrings in Camelot follow this format. Moreover, following the format will make sure that the API documentation is generated flawlessly.

- Make sure your commit messages follow [the seven rules of a great git commit message](https://chris.beams.io/posts/git-commit/):

- Separate subject from body with a blank line

- Limit the subject line to 50 characters

- Capitalize the subject line

- Do not end the subject line with a period

- Use the imperative mood in the subject line

- Wrap the body at 72 characters

- Use the body to explain what and why vs. how

- Please prefix the title of your pull request with [MRG] (Ready for Merge), if the contribution is complete and ready for a detailed review. An incomplete pull request's title should be prefixed with [WIP] (to indicate a work in progress), and changed to [MRG] when it's complete. A good [task list](https://blog.github.com/2013-01-09-task-lists-in-gfm-issues-pulls-comments/) in the PR description will ensure that other people get a better idea of what it proposes to do, which will also increase collaboration.

- If contributing new functionality, make sure that you add a unit test for it, while making sure that all previous tests pass. Camelot uses [pytest](https://docs.pytest.org/en/latest/) for testing. Tests can be run using:

<pre>

$ python setup.py test

</pre>

## Writing Documentation

Writing documentation, function docstrings, examples and tutorials is a great way to start contributing to open-source software! The documentation is present inside the `docs/` directory of the project repository.

It is written in [reStructuredText](https://en.wikipedia.org/wiki/ReStructuredText), with [Sphinx](http://www.sphinx-doc.org/en/master/) used to generate these lovely HTML files that you're currently reading (unless you're reading this on GitHub). You can edit the documentation using any text editor and then generate the HTML output by running `make html` in the `docs/` directory.

The function docstrings are written using the [numpydoc](https://numpydoc.readthedocs.io/en/latest/format.html) extension for Sphinx. Make sure you check out its format guidelines before you start writing one.

## Filing Issues

We use [GitHub issues](https://github.com/camelot-dev/camelot/issues) to keep track of all issues and pull requests. Before opening an issue (which asks a question or reports a bug), please use GitHub search to look for existing issues (both open and closed) that may be similar.

### Questions

Please don't use GitHub issues for support questions. A better place for them would be [Stack Overflow](http://stackoverflow.com). Make sure you tag them using the `python-camelot` tag.

### Bug Reports

In bug reports, make sure you include:

- Your operating system type and Python version number, along with the version numbers of NumPy, OpenCV and Camelot. You can use the following code snippet to find this information:

<pre>

import platform; print(platform.platform())

import sys; print('Python', sys.version)

import numpy; print('NumPy', numpy.\_\_version\_\_)

import cv2; print('OpenCV', cv2.\_\_version\_\_)

import camelot; print('Camelot', camelot.\_\_version\_\_)

</pre>

- The complete traceback. Just adding the exception message or a part of the traceback won't help us fix your issue sooner.

- Steps to reproduce the bug, using code snippets. See [Creating and highlighting code blocks](https://help.github.com/articles/creating-and-highlighting-code-blocks/).

- A link to the PDF document that you were trying to extract tables from, telling us what you expected the code to do and what actually happened.