# Contribution Guidelines

It’s such an honor to have you on board!

We are proud of this project and have been trying to make this project great since day one.

We believe you will love it, however, we do know that this is not a perfect project at all.

We have to

- implement features that make what you want to do possible and/or easily.

- write more examples that help you get familiar with Optuna.

- make issues and pull requests on GitHub fruitful.

- have more conversations and discussions on Gitter.

We need your help heartily, everything about Optuna you have in your mind push this project forward.

Join Us!

If you feel like giving your hand to us, here are some ways

- Implement a feature

- If you have some cool idea, please open an issue first to discuss design to make your idea in a better shape.

- Send a patch

- Dirty your hands by tackling [issues with `contribution-welcome` label](https://github.com/optuna/optuna/issues?q=is%3Aissue+is%3Aopen+label%3Acontribution-welcome)

- Report a bug

- If you find some bug, don't hesitate to report it! Your reports are really important!

- Fix/Improve documentation

- Documentation gets outdated easily, and can always be better, so feel free to fix & improve

- Let us & the Optuna community know your ideas, thought

- \_\_Contribution to Optuna includes not only sending pull requests, but also writing down your comments on issues and pull requests by others, and joining conversations/discussions on [Gitter](https://gitter.im/optuna/optuna).\_\_

- Also, sharing how you enjoy Optuna is a huge contribution! If you write some blog, let us know it!

If you choose to write some code, we have some conventions as follows.

- [Guidelines](#guidelines)

- [Unit Tests](#unit-tests)

- [Continuous Integration and Local Verification](#continuous-integration-and-local-verification)

- [Creating a Pull Request](#creating-a-pull-request)

## Guidelines

### Coding Style

Coding style is checked with [flake8](http://flake8.pycqa.org).

Additional conventions are described in the [Wiki](https://github.com/optuna/optuna/wiki/Coding-Style-Conventions).

### Documentation

When adding a new feature to the framework, you also need to document it in the reference.

The documentation source is stored under the [docs](./docs) directory and written in [reStructuredText format](http://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html).

To build the documentation, you need to run:

```bash

pip install -e ".[document]"

```

Then you can build the documentation in HTML format locally:

```bash

cd docs

make html

```

HTML files are generated under `build/html` directory. Open `index.html` with the browser and see

if it is rendered as expected.

### Type Hints

Type hints, [PEP484](https://www.python.org/dev/peps/pep-0484/), are checked with [mypy](http://mypy-lang.org/).

### Formatting

Code is formatted with [black](https://github.com/psf/black).

You have to install it first. This can be done with

`pip install black`. The command to format a certain file

is `black <filename\_with\_path> --line-length 99 --exclude="docs"`.

To just check the file you can use

`black <filename\_with\_path> --line-length 99 --check --exclude="docs"`.

You can also apply these operations to all files by replacing

`<filename\_with\_path>` with a simple `.`.

## Unit Tests

When adding a new feature or fixing a bug, you also need to write sufficient test code.

We use [pytest](https://pytest.org/) as the testing framework and

unit tests are stored under the [tests directory](./tests).

You can run all your tests as follows:

```bash

# Run all the unit tests.

pytest

# Run all the unit tests defined in the specified test file.

pytest tests/${TARGET\_TEST\_FILE\_NAME}

```

## Continuous Integration and Local Verification

CircleCI is used for continuous integration.

### Local Verification

By installing the [`circleci`](https://circleci.com/docs/2.0/local-cli/) local CLI and Docker, you can run tests locally.

```bash

circleci build --job <job\_name>

```

You can run the following jobs.

- `tests-python35`

- Runs unit tests under Python 3.5

- `tests-python36`

- Runs unit tests under Python 3.6

- `tests-python37`

- Runs unit tests under Python 3.7

- `tests-python38`

- Runs unit tests under Python 3.8

- `checks`

- Checks guidelines

- `document`

- Checks documentation build

- `doctest`

- Checks doctest validity

- `codecov`

- Checks unit test code coverage

#### Example

The following `circleci` job runs all unit tests in Python 3.7:

Note that this job will download several hundred megabytes of data to install all the packages required for testing, and take several tens of minutes to complete all tests.

```bash

circleci build --job tests-python37

```

## Creating a Pull Request

When you are ready to create a pull request, please try to keep the following in mind.

### Title

The title of your pull request should

- briefly describe and reflect the changes

- wrap any code with backticks

\*The title will be directly visible in the release notes.\*

#### Example

Introduces Tree-structured Parzen Estimator to `optuna.samplers`.

### Description

The description of your pull request should

- describe the motivation

- describe the changes

- if still work-in-progress, describe remaining tasks