Contributing to Mypy

====================

Welcome! Mypy is a community project that aims to work for a wide

range of Python users and Python codebases. If you're trying Mypy on

your Python code, your experience and what you can contribute are

important to the project's success.

Getting started, building, and testing

--------------------------------------

If you haven't already, take a look at the project's

[README.md file](README.md)

and the [Mypy documentation](http://mypy.readthedocs.io/en/latest/),

and try adding type annotations to your file and type-checking it with Mypy.

Discussion

----------

If you've run into behavior in Mypy you don't understand, or you're

having trouble working out a good way to apply it to your code, or

you've found a bug or would like a feature it doesn't have, we want to

hear from you!

Our main forum for discussion is the project's [GitHub issue

tracker](https://github.com/python/mypy/issues). This is the right

place to start a discussion of any of the above or most any other

topic concerning the project.

For less formal discussion we have a chat room on

[gitter.im](https://gitter.im/python/typing). Some Mypy core developers

are almost always present; feel free to find us there and we're happy

to chat. Substantive technical discussion will be directed to the

issue tracker.

(We also have an IRC channel, `#python-mypy` on irc.freenode.net.

This is lightly used, we have mostly switched to the gitter room

mentioned above.)

#### Code of Conduct

Everyone participating in the Mypy community, and in particular in our

issue tracker, pull requests, and IRC channel, is expected to treat

other people with respect and more generally to follow the guidelines

articulated in the [Python Community Code of

Conduct](https://www.python.org/psf/codeofconduct/).

First Time Contributors

-----------------------

Mypy appreciates your contribution! If you are interested in helping improve

mypy, there are several ways to get started:

\* Contributing to [typeshed](https://github.com/python/typeshed/issues) is a great way to

become familiar with Python's type syntax.

\* Work on [documentation issues](https://github.com/python/mypy/labels/documentation).

\* Ask on [the chat](https://gitter.im/python/typing) or on

[the issue tracker](https://github.com/python/mypy/issues) about good beginner issues.

Submitting Changes

------------------

Even more excellent than a good bug report is a fix for a bug, or the

implementation of a much-needed new feature. (\*) We'd love to have

your contributions.

(\*) If your new feature will be a lot of work, we recommend talking to

us early -- see below.

We use the usual GitHub pull-request flow, which may be familiar to

you if you've contributed to other projects on GitHub. For the mechanics,

see [our git and GitHub workflow help page](https://github.com/python/mypy/wiki/Using-Git-And-GitHub),

or [GitHub's own documentation](https://help.github.com/articles/using-pull-requests/).

Anyone interested in Mypy may review your code. One of the Mypy core

developers will merge your pull request when they think it's ready.

For every pull request, we aim to promptly either merge it or say why

it's not yet ready; if you go a few days without a reply, please feel

free to ping the thread by adding a new comment.

For a list of mypy core developers, see the file [CREDITS](CREDITS).

Preparing Changes

-----------------

Before you begin: if your change will be a significant amount of work

to write, we highly recommend starting by opening an issue laying out

what you want to do. That lets a conversation happen early in case

other contributors disagree with what you'd like to do or have ideas

that will help you do it.

The best pull requests are focused, clearly describe what they're for

and why they're correct, and contain tests for whatever changes they

make to the code's behavior. As a bonus these are easiest for someone

to review, which helps your pull request get merged quickly! Standard

advice about good pull requests for open-source projects applies; we

have [our own writeup](https://github.com/python/mypy/wiki/Good-Pull-Request)

of this advice.

See also our [coding conventions](https://github.com/python/mypy/wiki/Code-Conventions) --

which consist mainly of a reference to

[PEP 8](https://www.python.org/dev/peps/pep-0008/) -- for the code you

put in the pull request.

Also, do not squash your commits after you have submitted a pull request, as this

erases context during review. We will squash commits when the pull request is merged.

You may also find other pages in the

[Mypy developer guide](https://github.com/python/mypy/wiki/Developer-Guides)

helpful in developing your change.

Core developer guidelines

-------------------------

Core developers should follow these rules when processing pull requests:

\* Always wait for tests to pass before merging PRs.

\* Use "[Squash and merge](https://github.com/blog/2141-squash-your-commits)"

to merge PRs.

\* Delete branches for merged PRs (by core devs pushing to the main repo).

\* Edit the final commit message before merging to conform to the following

style (we wish to have a clean `git log` output):

\* When merging a multi-commit PR make sure that the commit message doesn't

contain the local history from the committer and the review history from

the PR. Edit the message to only describe the end state of the PR.

\* Make sure there is a \*single\* newline at the end of the commit message.

This way there is a single empty line between commits in `git log`

output.

\* Split lines as needed so that the maximum line length of the commit

message is under 80 characters, including the subject line.

\* Capitalize the subject and each paragraph.

\* Make sure that the subject of the commit message has no trailing dot.

\* Use the imperative mood in the subject line (e.g. "Fix typo in README").

\* If the PR fixes an issue, make sure something like "Fixes #xxx." occurs

in the body of the message (not in the subject).

\* Use Markdown for formatting.

Issue-tracker conventions

-------------------------

We aim to reply to all new issues promptly. We'll assign a milestone

to help us track which issues we intend to get to when, and may apply

labels to carry some other information. Here's what our milestones

and labels mean.

### Task priority and sizing

We use GitHub "labels" ([see our

list](https://github.com/python/mypy/labels)) to roughly order what we

want to do soon and less soon. There's two dimensions taken into

account: \*\*priority\*\* (does it matter to our users) and \*\*size\*\* (how

long will it take to complete).

Bugs that aren't a huge deal but do matter to users and don't seem

like a lot of work to fix generally will be dealt with sooner; things

that will take longer may go further out.

We are trying to keep the backlog at a manageable size, an issue that is

unlikely to be acted upon in foreseeable future is going to be

respectfully closed. This doesn't mean the issue is not important, but

rather reflects the limits of the team.

The \*\*question\*\* label is for issue threads where a user is asking a

question but it isn't yet clear that it represents something to actually

change. We use the issue tracker as the preferred venue for such

questions, even when they aren't literally issues, to keep down the

number of distinct discussion venues anyone needs to track. These might

evolve into a bug or feature request.

Issues \*\*without a priority or size\*\* haven't been triaged. We aim to

triage all new issues promptly, but there are some issues from previous

years that we haven't yet re-reviewed since adopting these conventions.

### Other labels

\* \*\*needs discussion\*\*: This issue needs agreement on some kind of

design before it makes sense to implement it, and it either doesn't

yet have a design or doesn't yet have agreement on one.

\* \*\*feature\*\*, \*\*bug\*\*, \*\*crash\*\*, \*\*refactoring\*\*, \*\*documentation\*\*:

These classify the user-facing impact of the change. Specifically

"refactoring" means there should be no user-facing effect.

\* \*\*topic-\*\* labels group issues touching a similar aspect of the

project, for example PEP 484 compatibility, a specific command-line

option or dependency.