.. \_hacking:

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

Ways to Contribute

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

This document aims to give an overview of the ways to contribute to SciPy. It

tries to answer commonly asked questions and provide some insight into how the

community process works in practice. Readers who are familiar with the SciPy

community and are experienced Python coders may want to jump straight to the

:ref:`contributor-toc`.

There are a lot of ways you can contribute:

- Contributing new code

- Fixing bugs, improving documentation, and other maintenance work

- Reviewing open pull requests

- Triaging issues

- Working on the `scipy.org`\_ website

- Answering questions and participating on the scipy-dev and scipy-user

`mailing lists`\_.

Contributing new code

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

If you have been working with the scientific Python toolstack for a while, you

probably have some code lying around of which you think "this could be useful

for others too". Perhaps it's a good idea then to contribute it to SciPy or

another open source project. The first question to ask is then, where does

this code belong? That question is hard to answer here, so we start with a

more specific one: \*what code is suitable for putting into SciPy?\*

Almost all of the new code added to SciPy has in common that it's potentially

useful in multiple scientific domains and it fits in the scope of existing

SciPy subpackages (see :ref:`deciding-on-new-features`). In principle new

subpackages can be added too, but this is far less common. For code that is

specific to a single application, there may be an existing project that can

use the code. Some SciKits (`scikit-learn`\_, `scikit-image`\_, `statsmodels`\_,

etc.) are good examples here; they have a narrower focus and because of that

more domain-specific code than SciPy.

Now if you have code that you would like to see included in SciPy, how do you

go about it? After checking that your code can be distributed in SciPy under a

compatible license (see :ref:`license-considerations`), the first step is to

discuss on the scipy-dev mailing list. All new features, as well as changes to

existing code, are discussed and decided on there. You can, and probably

should, already start this discussion before your code is finished. Remember

that in order to be added to SciPy your code will need to be reviewed by

someone else, so try to find someone willing to review your work while you're

at it.

Assuming the outcome of the discussion on the mailing list is positive and you

have a function or piece of code that does what you need it to do, what next?

Before code is added to SciPy, it at least has to have good documentation, unit

tests, benchmarks, and correct code style.

1. Unit tests

In principle you should aim to create unit tests that exercise all the code

that you are adding. This gives some degree of confidence that your code

runs correctly, also on Python versions and hardware or OSes that you don't

have available yourself. An extensive description of how to write unit

tests is given in :doc:`numpy:reference/testing`, and :ref:`runtests`

documents how to run them.

2. Benchmarks

Unit tests check for correct functionality; benchmarks measure code

performance. Not all existing SciPy code has benchmarks, but it should:

as SciPy grows it is increasingly important to monitor execution times in

order to catch unexpected regressions. More information about writing

and running benchmarks is available in :ref:`benchmarking-with-asv`.

3. Documentation

Clear and complete documentation is essential in order for users to be able

to find and understand the code. Documentation for individual functions

and classes -- which includes at least a basic description, type and

meaning of all parameters and returns values, and usage examples in

`doctest`\_ format -- is put in docstrings. Those docstrings can be read

within the interpreter, and are compiled into a reference guide in html and

pdf format. Higher-level documentation for key (areas of) functionality is

provided in tutorial format and/or in module docstrings. A guide on how to

write documentation is given in :ref:`numpy:howto-document`, and

:ref:`rendering-documentation` explains how to preview the documentation

as it will appear online.

4. Code style

Uniformity of style in which code is written is important to others trying

to understand the code. SciPy follows the standard Python guidelines for

code style, `PEP8`\_. In order to check that your code conforms to PEP8,

you can use the `pep8 package`\_ style checker. Most IDEs and text editors

have settings that can help you follow PEP8, for example by translating

tabs by four spaces. Using `pyflakes`\_ to check your code is also a good

idea. More information is available in :ref:`pep8-scipy`.

A :ref:`checklist<pr-checklist>`, including these and other requirements, is

available at the end of the example :ref:`development-workflow`.

Another question you may have is: \*where exactly do I put my code\*? To answer

this, it is useful to understand how the SciPy public API (application

programming interface) is defined. For most modules the API is two levels

deep, which means your new function should appear as

``scipy.subpackage.my\_new\_func``. ``my\_new\_func`` can be put in an existing or

new file under ``/scipy/<subpackage>/``, its name is added to the ``\_\_all\_\_``

list in that file (which lists all public functions in the file), and those

public functions are then imported in ``/scipy/<subpackage>/\_\_init\_\_.py``. Any

private functions/classes should have a leading underscore (``\_``) in their

name. A more detailed description of what the public API of SciPy is, is given

in :ref:`scipy-api`.

Once you think your code is ready for inclusion in SciPy, you can send a pull

request (PR) on Github. We won't go into the details of how to work with git

here, this is described well in :ref:`git-development`

and on the `Github help pages`\_. When you send the PR for a new

feature, be sure to also mention this on the scipy-dev mailing list. This can

prompt interested people to help review your PR. Assuming that you already got

positive feedback before on the general idea of your code/feature, the purpose

of the code review is to ensure that the code is correct, efficient and meets

the requirements outlined above. In many cases the code review happens

relatively quickly, but it's possible that it stalls. If you have addressed

all feedback already given, it's perfectly fine to ask on the mailing list

again for review (after a reasonable amount of time, say a couple of weeks, has

passed). Once the review is completed, the PR is merged into the "master"

branch of SciPy.

The above describes the requirements and process for adding code to SciPy. It

doesn't yet answer the question though how decisions are made exactly. The

basic answer is: decisions are made by consensus, by everyone who chooses to

participate in the discussion on the mailing list. This includes developers,

other users and yourself. Aiming for consensus in the discussion is important

-- SciPy is a project by and for the scientific Python community. In those

rare cases that agreement cannot be reached, the maintainers of the module

in question can decide the issue.

.. \_license-considerations:

License Considerations

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

\*I based my code on existing Matlab/R/... code I found online, is this OK?\*

It depends. SciPy is distributed under a BSD license, so if the code that you

based your code on is also BSD licensed or has a BSD-compatible license (e.g.

MIT, PSF) then it's OK. Code which is GPL or Apache licensed, has no

clear license, requires citation or is free for academic use only can't be

included in SciPy. Therefore if you copied existing code with such a license

or made a direct translation to Python of it, your code can't be included.

If you're unsure, please ask on the scipy-dev `mailing list <mailing lists>`\_.

\*Why is SciPy under the BSD license and not, say, the GPL?\*

Like Python, SciPy uses a "permissive" open source license, which allows

proprietary re-use. While this allows companies to use and modify the software

without giving anything back, it is felt that the larger user base results in

more contributions overall, and companies often publish their modifications

anyway, without being required to. See John Hunter's `BSD pitch`\_.

For more information about SciPy's license, see :ref:`scipy-licensing`.

Maintaining existing code

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

The previous section talked specifically about adding new functionality to

SciPy. A large part of that discussion also applies to maintenance of existing

code. Maintenance means fixing bugs, improving code quality, documenting

existing functionality better, adding missing unit tests, adding performance

benchmarks, keeping build scripts up-to-date, etc. The SciPy `issue list`\_

contains all reported bugs, build/documentation issues, etc. Fixing issues

helps improve the overall quality of SciPy, and is also a good way

of getting familiar with the project. You may also want to fix a bug because

you ran into it and need the function in question to work correctly.

The discussion on code style and unit testing above applies equally to bug

fixes. It is usually best to start by writing a unit test that shows the

problem, i.e. it should pass but doesn't. Once you have that, you can fix the

code so that the test does pass. That should be enough to send a PR for this

issue. Unlike when adding new code, discussing this on the mailing list may

not be necessary - if the old behavior of the code is clearly incorrect, no one

will object to having it fixed. It may be necessary to add some warning or

deprecation message for the changed behavior. This should be part of the

review process.

.. note::

Pull requests that \*only\* change code style, e.g. fixing some PEP8 issues in

a file, are discouraged. Such PRs are often not worth cluttering the git

annotate history, and take reviewer time that may be better spent in other ways.

Code style cleanups of code that is touched as part of a functional change

are fine however.

Reviewing pull requests

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

Reviewing open pull requests (PRs) is very welcome, and a valuable way to help

increase the speed at which the project moves forward. If you have specific

knowledge/experience in a particular area (say "optimization algorithms" or

"special functions") then reviewing PRs in that area is especially valuable -

sometimes PRs with technical code have to wait for a long time to get merged

due to a shortage of appropriate reviewers.

We encourage everyone to get involved in the review process; it's also a

great way to get familiar with the code base. Reviewers should ask

themselves some or all of the following questions:

- Was this change adequately discussed (relevant for new features and changes

in existing behavior)?

- Is the feature scientifically sound? Algorithms may be known to work based on

literature; otherwise, closer look at correctness is valuable.

- Is the intended behavior clear under all conditions (e.g. unexpected inputs

like empty arrays or nan/inf values)?

- Does the code meet the quality, test and documentation expectation outline

under `Contributing new code`\_?

If we do not know you yet, consider introducing yourself.

Other ways to contribute

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

There are many ways to contribute other than writing code.

Triaging issues (investigating bug reports for validity and possible actions to

take) is also a useful activity. SciPy has many hundreds of open issues;

closing invalid ones and correctly labeling valid ones (ideally with some first

thoughts in a comment) allows prioritizing maintenance work and finding related

issues easily when working on an existing function or subpackage.

Participating in discussions on the scipy-user and scipy-dev `mailing lists`\_ is

a contribution in itself. Everyone who writes to those lists with a problem or

an idea would like to get responses, and writing such responses makes the

project and community function better and appear more welcoming.

The `scipy.org`\_ website contains a lot of information on both SciPy the

project and SciPy the community, and it can always use a new pair of hands.

The sources for the website live in their own separate repo:

https://github.com/scipy/scipy.org

Getting started

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

Thanks for your interest in contributing to SciPy! If you're interested in

contributing code, we hope you'll continue on to the :ref:`contributor-toc`

for details on how to set up your development environment, implement your

improvements, and submit your first PR!

.. \_scikit-learn: http://scikit-learn.org

.. \_scikit-image: http://scikit-image.org/

.. \_statsmodels: https://www.statsmodels.org/

.. \_testing guidelines: https://docs.scipy.org/doc/numpy/reference/testing.html

.. \_formatted correctly: https://docs.scipy.org/doc/numpy/dev/gitwash/development\_workflow.html#writing-the-commit-message

.. \_bug report: https://scipy.org/bug-report.html

.. \_PEP8: https://www.python.org/dev/peps/pep-0008/

.. \_pep8 package: https://pypi.python.org/pypi/pep8

.. \_pyflakes: https://pypi.python.org/pypi/pyflakes

.. \_Github help pages: https://help.github.com/articles/set-up-git/

.. \_issue list: https://github.com/scipy/scipy/issues

.. \_Github: https://github.com/scipy/scipy

.. \_scipy.org: https://scipy.org/

.. \_scipy.github.com: https://scipy.github.com/

.. \_scipy.org-new: https://github.com/scipy/scipy.org-new

.. \_documentation wiki: https://docs.scipy.org/scipy/Front%20Page/

.. \_SciPy Central: https://web.archive.org/web/20170520065729/http://central.scipy.org/

.. \_doctest: https://pymotw.com/3/doctest/

.. \_virtualenv: https://virtualenv.pypa.io/

.. \_virtualenvwrapper: https://bitbucket.org/dhellmann/virtualenvwrapper/

.. \_bsd pitch: http://nipy.sourceforge.net/nipy/stable/faq/johns\_bsd\_pitch.html

.. \_Pytest: https://pytest.org/

.. \_mailing lists: https://www.scipy.org/scipylib/mailing-lists.html

.. \_Spyder: https://www.spyder-ide.org/

.. \_Anaconda SciPy Dev Part I (macOS): https://youtu.be/1rPOSNd0ULI

.. \_Anaconda SciPy Dev Part II (macOS): https://youtu.be/Faz29u5xIZc

.. \_SciPy Development Workflow: https://youtu.be/HgU01gJbzMY