You don’t have to be a compiler engineer to work on Python

Savannah Bailey, EuroPython 2025

Hey, I’m Savannah! 👋

💙 Python Core Developer and PEP author

🧪 Jupyter Foundation Governing Board Treasurer

❄️ Python at Snowflake

✨ (Mostly) self-taught developer

🐈 Cat mom of three

Let me take you back to 2020

No, no…not that part

I was learning bits and pieces about how Python worked.

“Let’s be real, that’s never going to happen”

“You don’t even know C”

“It would be so cool to contribute to Python”

“Maybe someday but probably not”

Me, about five years ago

“You don’t even work as an engineer anymore”

Good news!

I was **wrong** 🤗

I want to give you a bit of a PEP talk ❤️

You probably already have relevant skills!

🧪 Do you know how to test or debug Python code?

📚 Do you like to write docs or share examples?

🧰 Have you worked in other areas of computing?

🧩 Or, do you just want to help untangle issues?

Contributions are more than just commits

Using triaging to onboard

What is triaging?

- Reproducing issues
- Finding minimum reproducers
- Helping categorize issues
- Suggesting potential fixes

Why it’s valuable

- Immensely helpful to maintainers
- Improves repository hygiene
- Streamlines the fix process
- Perfect on-ramp for contributors

Docs are important infrastructure

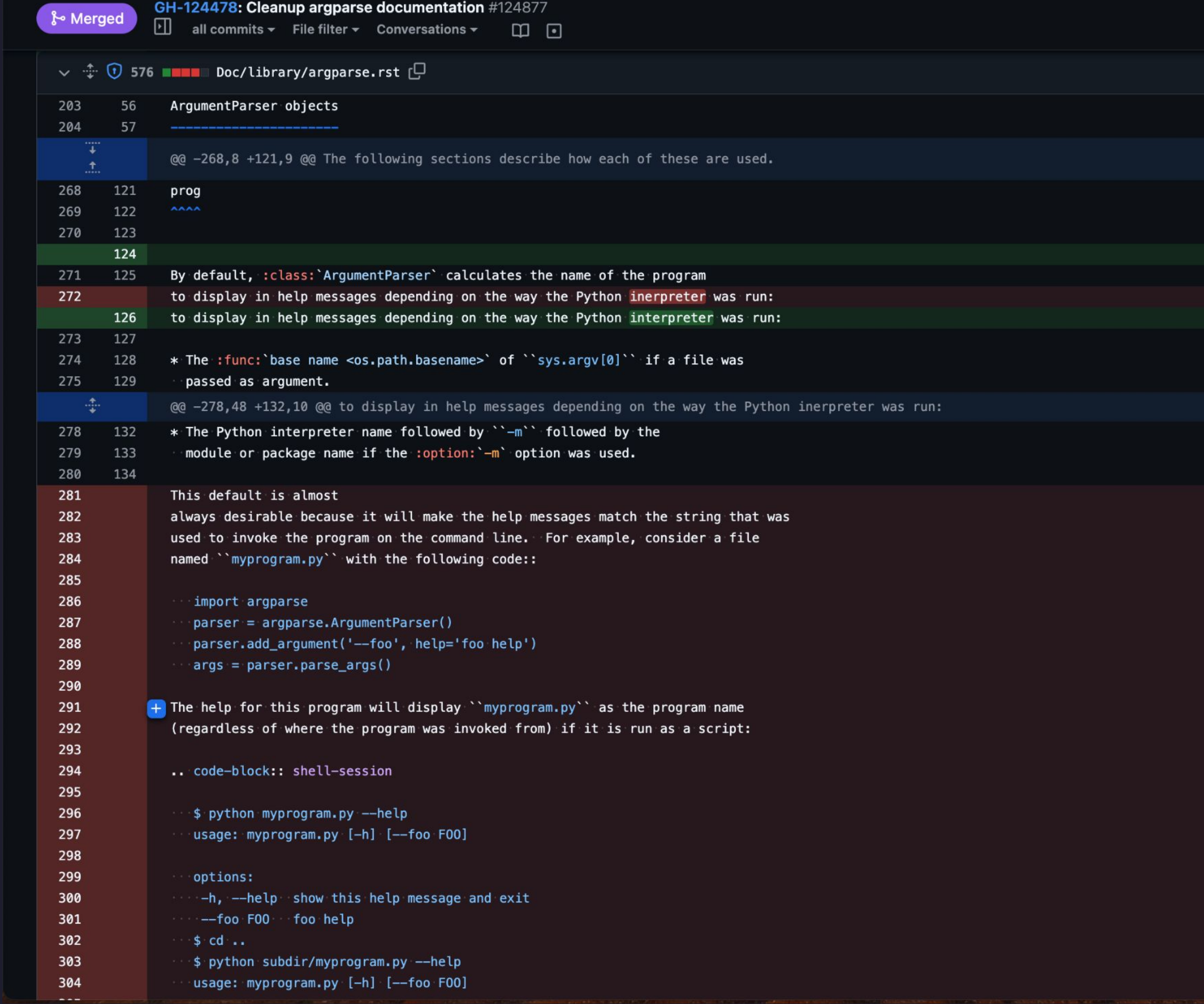
Why documentation matters

- Often the first interaction users have with Python
- Critical for adoption & learning curve
- Makes Python more accessible

How new contributors can help

- Fix inconsistencies and typos
- Write beginner-friendly explanations
- Improve examples
- Help with translations

…typos, restructuring and examples, oh my!



GH-124478: Cleanup argparse documentation #124877

576 Doc/library/argparse.rst

```
ArgumentParser objects
----------------------

@@ -268,8 +121,9 @@ The following sections describe how each of these are used.

prog
^^^^


By default, :class:`ArgumentParser` calculates the name of the program
to display in help messages depending on the way the Python inerpreter was run:
to display in help messages depending on the way the Python interpreter was run:

* The :func:`base name <os.path.basename>` of ``sys.argv[0]`` if a file was
  passed as argument.
@@ -278,48 +132,10 @@ to display in help messages depending on the way the Python inerpreter was run:
* The Python interpreter name followed by ``-m`` followed by the
  module or package name if the :option:`-m` option was used.

This default is almost
always desirable because it will make the help messages match the string that was
used to invoke the program on the command line.  For example, consider a file
named ``myprogram.py`` with the following code::

   import argparse
   parser = argparse.ArgumentParser()
   parser.add_argument('--foo', help='foo help')
   args = parser.parse_args()

The help for this program will display ``myprogram.py`` as the program name
(regardless of where the program was invoked from) if it is run as a script:

.. code-block:: shell-session

   $ python myprogram.py --help
   usage: myprogram.py [-h] [--foo FOO]

   options:
    -h, --help  show this help message and exit
    --foo FOO   foo help
   $ cd ..
   $ python subdir/myprogram.py --help
   usage: myprogram.py [-h] [--foo FOO]
```

The standard library is (mostly) Python

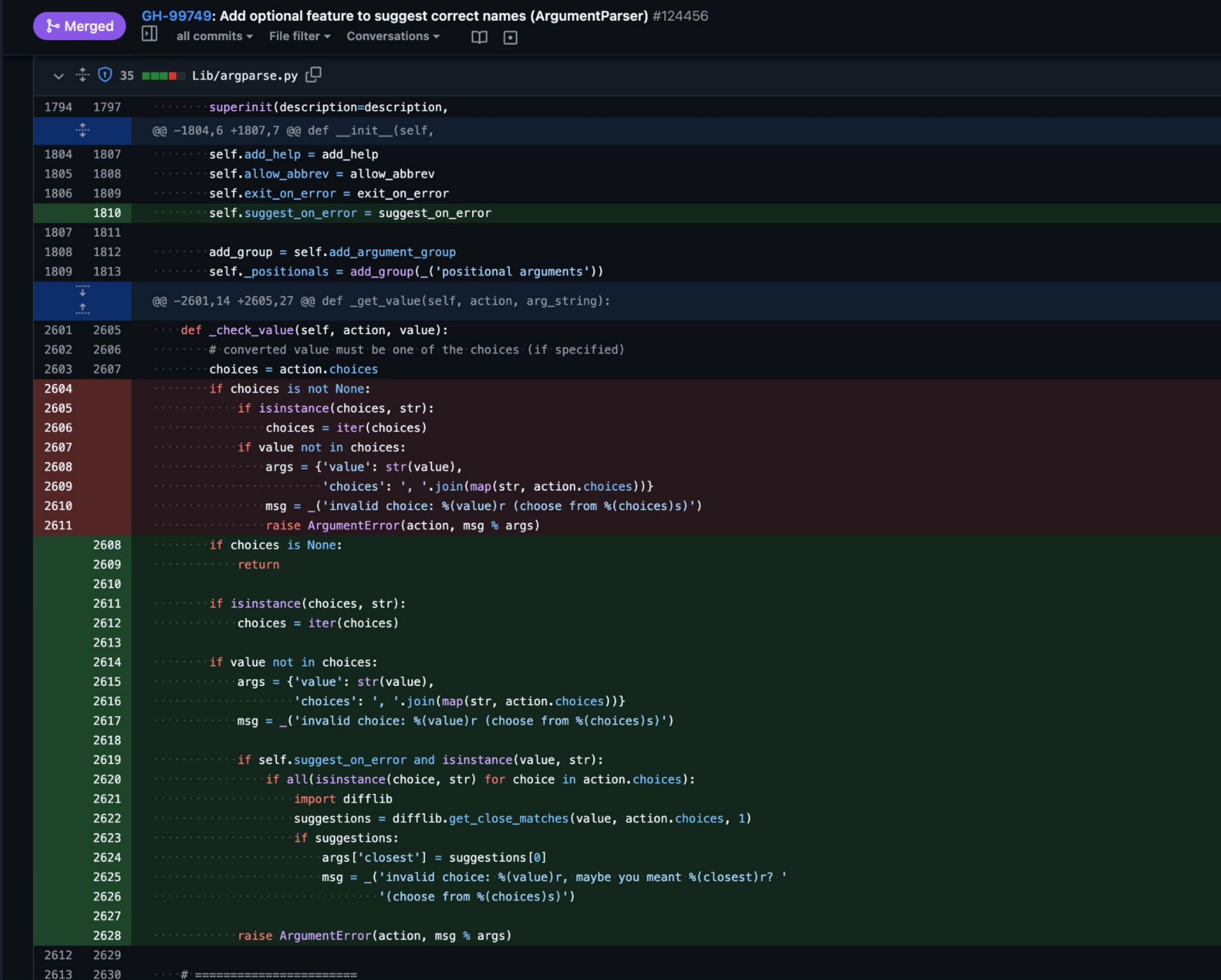
Considerations

- Start small and build expertise
- Bugs can be features - a lesson in backward compatibility

How can I help?

- Issue labels…
- Triage! Triage! Triage!
- Improve test coverage

…Just Python


Merged GH-99749: Add optional feature to suggest correct names (ArgumentParser) #124456

all commits ▾ File filter ▾ Conversations ▾

35 Lib/argparse.py

```
        superinit(description=description,
@@ -1804,6 +1807,7 @@ def __init__(self,
        self.add_help = add_help
        self.allow_abbrev = allow_abbrev
        self.exit_on_error = exit_on_error
        self.suggest_on_error = suggest_on_error

        add_group = self.add_argument_group
        self._positionals = add_group(_('positional arguments'))
@@ -2601,14 +2605,27 @@ def _get_value(self, action, arg_string):
    def _check_value(self, action, value):
        # converted value must be one of the choices (if specified)
        choices = action.choices
        if choices is not None:
            if isinstance(choices, str):
                choices = iter(choices)
            if value not in choices:
                args = {'value': str(value),
                        'choices': ', '.join(map(str, action.choices))}
                msg = _('invalid choice: %(value)r (choose from %(choices)s)')
                raise ArgumentError(action, msg % args)
        if choices is None:
            return

        if isinstance(choices, str):
            choices = iter(choices)

        if value not in choices:
            args = {'value': str(value),
                    'choices': ', '.join(map(str, action.choices))}
            msg = _('invalid choice: %(value)r (choose from %(choices)s)')

            if self.suggest_on_error and isinstance(value, str):
                if all(isinstance(choice, str) for choice in action.choices):
                    import difflib
                    suggestions = difflib.get_close_matches(value, action.choices, 1)
                    if suggestions:
                        args['closest'] = suggestions[0]
                        msg = _('invalid choice: %(value)r, maybe you meant %(closest)r? '
                                '(choose from %(choices)s)')

            raise ArgumentError(action, msg % args)

    # ======================
```

You don’t have to know the ins and outs of the interpreter to contribute to Python*

* But you can learn if you’re interested!

DevOps was my gateway to compiler engineering

Working on the JIT

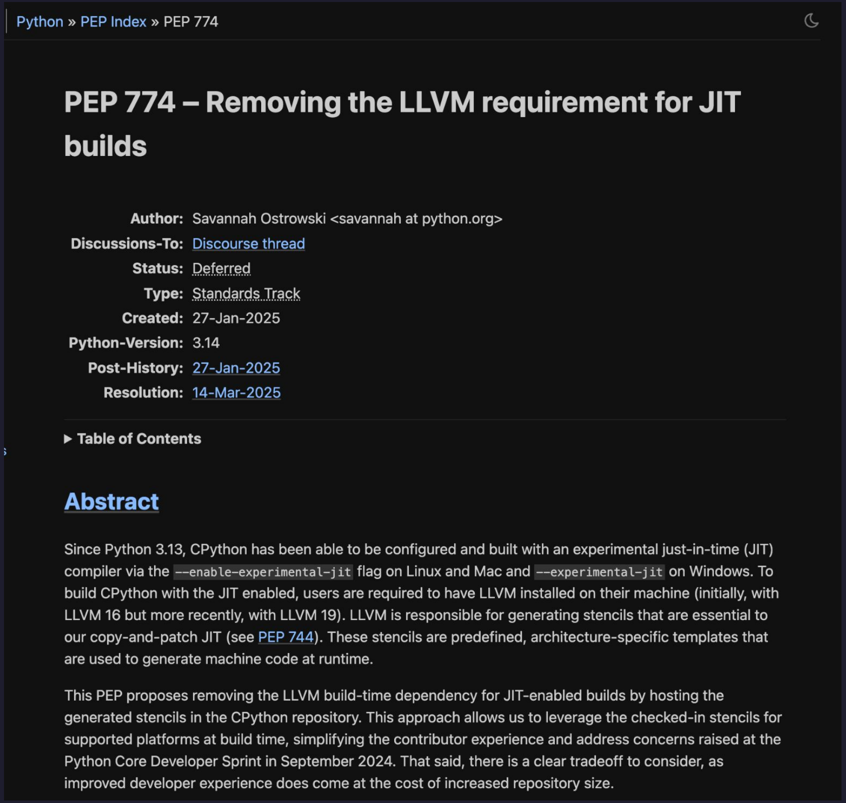
What I brought with me

- DevOps background: CI/CD, containers, build systems
- Experience wrangling GitHub Actions, multi-arch builds
- Not afraid of platform quirks and dependency hell

Where that helped

- Dependency upgrades for our compiler toolchain
- macOS multi-arch builds
- CI workflows improvements that JIT testing and development easier

…Just DevOps (and more Python)



Python » PEP Index » PEP 774

PEP 774 – Removing the LLVM requirement for JIT builds

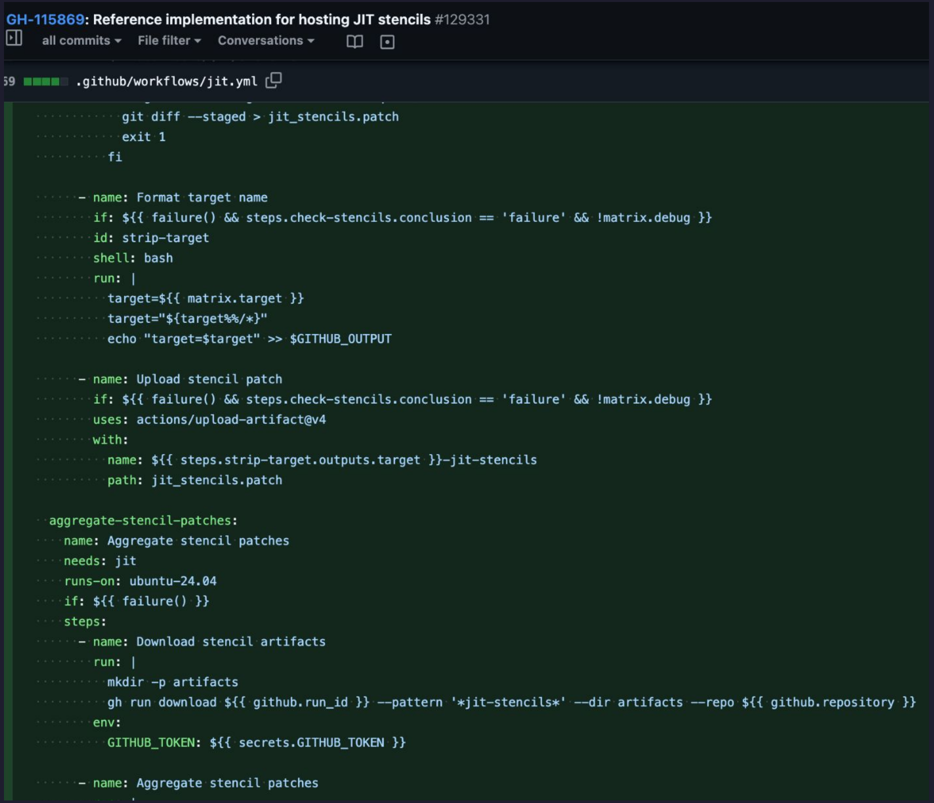
Author:	Savannah Ostrowski <savannah at python.org>
Discussions-To:	Discourse thread
Status:	Deferred
Type:	Standards Track
Created:	27-Jan-2025
Python-Version:	3.14
Post-History:	27-Jan-2025
Resolution:	14-Mar-2025

▶ **Table of Contents**

Abstract

Since Python 3.13, CPython has been able to be configured and built with an experimental just-in-time (JIT) compiler via the `--enable-experimental-jit` flag on Linux and Mac and `--experimental-jit` on Windows. To build CPython with the JIT enabled, users are required to have LLVM installed on their machine (initially, with LLVM 16 but more recently, with LLVM 19). LLVM is responsible for generating stencils that are essential to our copy-and-patch JIT (see PEP 744). These stencils are predefined, architecture-specific templates that are used to generate machine code at runtime.

This PEP proposes removing the LLVM build-time dependency for JIT-enabled builds by hosting the generated stencils in the CPython repository. This approach allows us to leverage the checked-in stencils for supported platforms at build time, simplifying the contributor experience and address concerns raised at the Python Core Developer Sprint in September 2024. That said, there is a clear tradeoff to consider, as improved developer experience does come at the cost of increased repository size.



GH-115869: Reference implementation for hosting JIT stencils #129331

59 .github/workflows/jit.yml

```
            git diff --staged > jit_stencils.patch
            exit 1
          fi

      - name: Format target name
        if: ${{ failure() && steps.check-stencils.conclusion == 'failure' && !matrix.debug }}
        id: strip-target
        shell: bash
        run: |
          target=${{ matrix.target }}
          target="${target%%/*}"
          echo "target=$target" >> $GITHUB_OUTPUT

      - name: Upload stencil patch
        if: ${{ failure() && steps.check-stencils.conclusion == 'failure' && !matrix.debug }}
        uses: actions/upload-artifact@v4
        with:
          name: ${{ steps.strip-target.outputs.target }}-jit-stencils
          path: jit_stencils.patch

  aggregate-stencil-patches:
    name: Aggregate stencil patches
    needs: jit
    runs-on: ubuntu-24.04
    if: ${{ failure() }}
    steps:
      - name: Download stencil artifacts
        run: |
          mkdir -p artifacts
          gh run download ${{ github.run_id }} --pattern '*jit-stencils*' --dir artifacts --repo ${{ github.repository }}
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

      - name: Aggregate stencil patches
```

I left out a very important detail…

Community 🌳

“Have <u>you</u> ever thought about becoming a core developer?”

Find your community 🤝

Each contributor brings something new to Python

Do it scared ✨

Contribution Toolkit

🐍	github.com/python/cpython	The home of Python’s code. Follow issues, submit PRs, or just watch the repo to learn.
📝	peps.python.org	Docs proposing/explaining major Python changes — how and why things work the way they do.
💬	discuss.python.org	Where high-level discussion happens on features, governance, packaging, and ideas.
📃	devguide.python.org	Everything you need to start contributing: setup, tools, triage process, testing, and more.

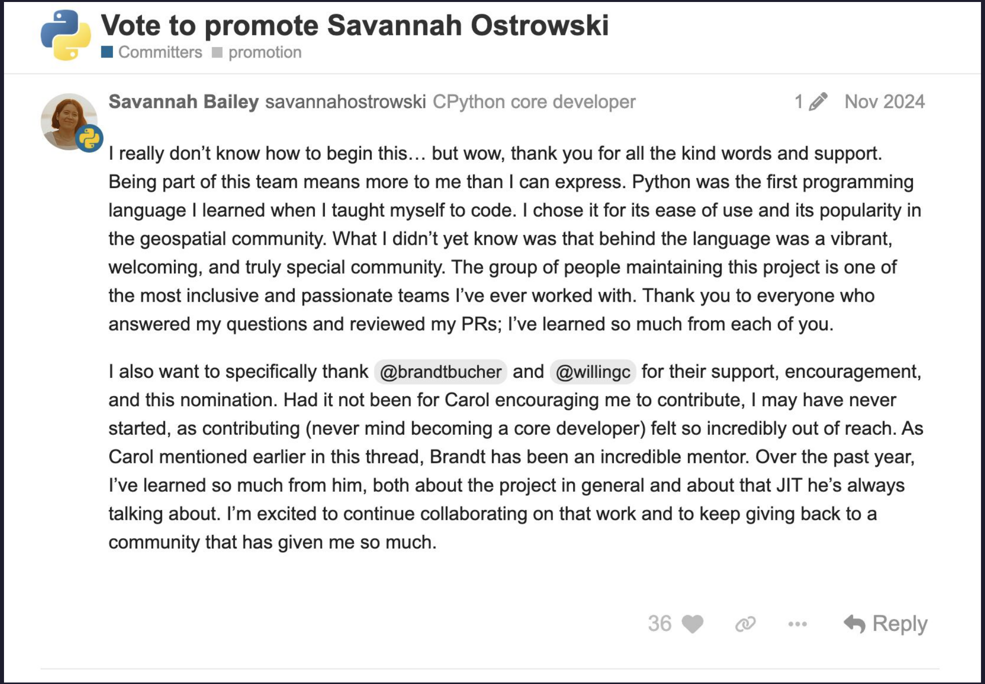
Thank you ❤️

You can find me many places on the Internet

Bluesky: @savannah.dev

GitHub: @savannahostrowski

Email: savannah@python.org



Vote to promote Savannah Ostrowski

Committers promotion

Savannah Bailey savannahostrowski CPython core developer 1 Nov 2024

I really don't know how to begin this… but wow, thank you for all the kind words and support. Being part of this team means more to me than I can express. Python was the first programming language I learned when I taught myself to code. I chose it for its ease of use and its popularity in the geospatial community. What I didn't yet know was that behind the language was a vibrant, welcoming, and truly special community. The group of people maintaining this project is one of the most inclusive and passionate teams I've ever worked with. Thank you to everyone who answered my questions and reviewed my PRs; I've learned so much from each of you.

I also want to specifically thank @brandtbucher and @willingc for their support, encouragement, and this nomination. Had it not been for Carol encouraging me to contribute, I may have never started, as contributing (never mind becoming a core developer) felt so incredibly out of reach. As Carol mentioned earlier in this thread, Brandt has been an incredible mentor. Over the past year, I've learned so much from him, both about the project in general and about that JIT he's always talking about. I'm excited to continue collaborating on that work and to keep giving back to a community that has given me so much.