# Contributing to KaTeX

We welcome pull requests to KaTeX. If you'd like to add a new symbol, or try to

tackle adding a larger feature, keep reading. If you have any questions, or want

help solving a problem, feel free to stop by our [gitter channel](https://gitter.im/KaTeX/KaTeX).

## Helpful contributions

If you'd like to contribute, try contributing new symbols or functions that

KaTeX doesn't currently support. The documentation has pages listing

[supported functions](https://katex.org/docs/supported.html) and

[functions that KaTeX supports and some that it doesn't support](https://katex.org/docs/support\_table.html).

You can check them to see if we don't support a function you like, or try your

function in the interactive demo at [http://katex.org/](http://katex.org/).

The wiki has a page that describes how to [examine TeX commands and where to find

rules](https://github.com/KaTeX/KaTeX/wiki/Examining-TeX) which can be quite

useful when adding new commands.

#### Single symbols

There are many individual symbols that KaTeX doesn't yet support. Read through

the [symbols.js](src/symbols.js) file for more information on how to add a

symbol.

To figure out the unicode symbol for the symbol you are trying to add, try using

the symbol in MathJax to see what unicode symbol it outputs. An interactive

MathJax shell can be found [here](http://fiddle.jshell.net/YpqVp/41/show/).

To figure out what group your symbol falls into, look through the symbols list

to find other symbols of a similar kind. (e.g. if you were adding `\neq`, look

for `=`). If you cannot find anything similar, or are unsure, you can try using

your symbol in TeX surrounded by other different kinds of symbols, and seeing

whether your spacing matches the spacing that TeX produces.

Once your symbol works, check the JavaScript console to make sure you don't get

a message like "Can't find character metrics for \\_" when you render your symbol.

If you do, check out [extract\_ttfs.py](metrics/extract\_ttfs.py).

#### Adding new functions

New functions should be added in [src/functions](src/functions) using

`defineFunction` from [defineFunction.js](src/defineFunction.js). Read the

comments in this file to get started. Look at

[phantom.js](src/functions/phantom.js) and

[delimsizing.js](src/functions/delimsizing.js) as examples of how to use

`defineFunction`. Notice how delimsizing.js groups several related functions

together in a single call to `defineFunction`.

The new method of defining functions combines methods that were previously

spread out over three different files [functions.js](src/functions.js),

[buildHTML.js](src/buildHTML.js), [buildMathML.js](src/buildMathML.js) into a

single file. The goal is to have all functions use this new system.

#### Macros

Macros should be added in [src/macros.js](src/macros.js) using `defineMacro`.

They are expanded in the "gullet" (`MacroExpander`).

## Testing

Local testing can be done by running the webpack-dev-server using configuration

`webpack.dev.js`. Run `yarn` to install dependencies, and then `yarn start`

to start the server.

This will host an interactive editor at

[http://localhost:7936/](http://localhost:7936/) to play around with and test

changes.

#### Jest tests

The JavaScript parser and some of the HTML and MathML tree

builders are tested with Jest. These tests can be run using node with

`yarn test:jest`. If you need to debug the tests see

[https://facebook.github.io/jest/docs/troubleshooting.html](https://facebook.github.io/jest/docs/troubleshooting.html)

The interactive editor can also be used for debugging tests in the browser by

copy/pasting the test case to be debugged into the editor. The permalink option

can come in really useful when doing repeated runs of the same test case.

The Jest tests should be run after every change, even the addition of small

symbols. However, [CircleCI](https://circleci.com/gh/KaTeX/KaTeX) will run these

tests when you submit a pull request, in case you forget.

If you make any changes to Parser.js, add Jest tests to ensure they work.

Some tests verify the structure of the output tree using [snapshot testing](https://facebook.github.io/jest/docs/en/snapshot-testing.html).

Those snapshots can be updated by running `yarn test:jest:update`.

Also, test code coverage can be collected by `yarn test:jest:coverage`.

You can view the report in `coverage/lcov-report/index.html`.

#### Screenshot tests

To ensure the final output looks good, we screenshot different expressions.

These tests can be run by using the

[screenshotter docker](https://github.com/KaTeX/KaTeX/tree/master/dockers/screenshotter).

The screenshot tests should be run if you add anything more significant than

individual symbols. These tests are not automatically run, so please remember!

If the new images are different (meaning they are not byte-by-byte the same as

the old ones), inspect them visually. If there are no visible changes, that is

okay. If things change in a way consistent with your additions, explain what

changed and why. Otherwise, figure out what is causing the changes and fix it!

If you add a feature that is dependent on the final output looking the way you

created it, add a screenshot test. See

[ss\_data.yaml](test/screenshotter/ss\_data.yaml).

You can use our

[texcmp](https://github.com/KaTeX/KaTeX/tree/master/dockers/texcmp) tool

to compare the outputs of a screenshot test as generated by KaTeX and LaTeX.

It's often useful to attach the resulting "visual diff" to your pull request

with a new feature.

#### Testing in other browsers

KaTeX supports all major browsers, including IE 11 and newer. Unfortunately, it

is hard to test new changes in many browsers. If you can, please test your

changes in as many browsers as possible.

## Building

KaTeX is built using webpack with configuration `webpack.config.js`. Run

`yarn build` to build the project.

## Style guide

Code

- 4 spaces for indentation

- 80 character line length

- commas last

- declare variables in the outermost scope that they are used

- camelCase for variables in JavaScript

- snake\_case for variables in Python

In general, try to make your code blend in with the surrounding code.

The code can be linted by running `yarn test:lint`, which lints JavaScript

files using ESLint and stylesheets using stylelint. They must pass to commit

the changes.

Some files have flowtype annotations and can be checked for type errors using

Flow by running `yarn test:flow`. See [Flow](https://flow.org/) for more details.

## Pull Requests

- link back to the original issue(s) whenever possible

- new commands should be added to `docs/support\_table.md` and `docs/supported.md`

- commits should be squashed before merging

- large pull requests should be broken into separate pull requests (or multiple logically cohesive commits), if possible

## Working with submodules

The fonts for KaTeX live in a submodule stored in `submodules/katex-fonts`.

When you first clone the KaTeX repository, use

`git submodule update --init --recursive` to download the corresponding

fonts repository. After running `yarn`, you should have Git hooks that

will automatically run this command after switching to branches

where `submodules/katex-fonts` point to different commits.

When submitting pull requests that update katex-fonts, you'll need to submit

two pull requests: one for [KaTeX/katex-fonts](https://github.com/KaTeX/katex-fonts) and one for [KaTeX/KaTeX](https://github.com/KaTeX/KaTeX).

For more info about how to use git submodules,

see https://chrisjean.com/git-submodules-adding-using-removing-and-updating/.

## CLA

In order to contribute to KaTeX, you must first sign the CLA, found at www.khanacademy.org/r/cla

## License

KaTeX is licenced under the [MIT License](http://opensource.org/licenses/MIT).