Contents

Books with Jupyter

DOI 10.5281/zenodo.2561065

Jupyter Book is an open source project for building beautiful, publication-quality books and documents from computational material.

Here are some of the features of Jupyter Book:

✓ **Write publication-quality content in Markdown**
You can write in either Jupyter Markdown, or an extended flavor of Markdown with publishing features. This includes support for rich syntax such as citations and cross-references, math and equations, and figures.

✓ **Write content in Jupyter Notebook**
This allows you to include your code and outputs in your book. You can also write notebooks entirely in Markdown that get executed when you build your book.

✓ **Execute and cache your book's content**
For `.ipynb` and Markdown notebooks, execute code and insert the latest outputs into your book. In addition, cache and re-use outputs to be used later.

✓ **Insert notebook outputs into your content**
Generate outputs as you build your documentation, and insert them in-line with your content across pages.

✓ **Add interactivity to your book**
You can toggle cell visibility, include interactive outputs from Jupyter, and connect with online services like Binder.

✓ **Generate a variety of outputs**
This includes single- and multi-page websites, as well as PDF outputs.

✓ **Build books with a simple command-line interface**
You can quickly generate your books with one command, like so: `jupyter-book build mybook/`

This website is built with Jupyter Book! You can browse its contents to the left to see what is possible.

Learn more and get involved

Jupyter Book is an open community that welcomes your feedback, input, and contributions!

💡 **Open an issue**
to provide feedback and new ideas, and to help others.

📚 **See the Jupyter Book Gallery**
to be inspired by Jupyter Books that others have created.

👍 **Vote for new features**
by adding a +1 to issues you'd like to see completed.

🙌 **Contribute to Jupyter Book**
by following our contributing guidelines and finding an issue to work on. See the feature voting leaderboard for inspiration.

Install Jupyter Book

You can install Jupyter Book via `pip`:

```
pip install -U jupyter-book
```

This will install everything you need to build a Jupyter Book locally.

Get started

To get started with Jupyter Book, you can either

- check out the getting started guide,
- browse the contents of the navigation menu of this book (to the left, if you're on a laptop), or
- review the example project shown immediately below (if you like learning from examples).

Warning

Jupyter Book `0.8` is a total re-write from previous versions, and some things have changed. See the legacy upgrade guide for how to upgrade, and legacy.jupyterbook.org for legacy documentation.

In addition, note that Jupyter Book is pre-1.0. Its API may change!

To install `jupyter-book` from pip, run the following command:

```
pip install -U jupyter-book
```

Create a template book to get started right away, or see the getting started guide for more information.

A small example project

Here's a short example of a web-based book created by Jupyter Book.

Some of the features on display include

- Jupyter Notebook-style inputs and outputs
- citations
- numbered equations
- numbered figures with captions and cross-referencing

The source files can be found on GitHub in the docs directory. These files are written in MyST Markdown, an extension of the Jupyter Notebook Markdown, that allows for additional scientific markup. They could alternatively have been written directly as Jupyter notebooks.

Build the demo book

You can build this book locally on the command line via the following steps:

1. Ensure you have a recent version of Anaconda Python installed.
2. Clone the repository containing the demo book source files

   ```
   git clone https://github.com/executablebooks/quantecon-mini-example
   cd quantecon-mini-example
   ```

3. Install the Python libraries needed to run the code in this particular example from the `environment.yml` file. This includes the latest version of Jupyter Book:

   ```
   conda env create -f environment.yml
   conda activate qe-mini-example
   ```

4. Run Jupyter Book over the source files

   ```
   jupyter-book build ./mini_book
   ```

5. View the result through a browser — try (with, say, firefox)

   ```
   firefox mini_book/_build/html/index.html
   ```

   (or simply double-click on the `html` file)

If you'd like to install Jupyter Book with `pip`, you can do so with:

```
pip install -U jupyter-book
```

See the getting started page for more information.

Now you might like to try editing the files in `mini_book/docs` and then rebuilding.

Further reading

See the full QuantEcon example for a longer Jupyter Book use case, drawn from the same source material.

For more information on how to use Jupyter Book, see Overview.

Under the hood - the components of Jupyter Book

Jupyter Book is a wrapper around a collection of tools in the Python ecosystem that make it easier to publish computational documents. Here are a few key pieces:

- It uses the MyST Markdown language in Markdown and notebook documents. This allows users to write rich, publication-quality markup in their documents.
- It uses the MyST-NB package to parse and read-in notebooks so they are built into your book.
- It uses the Sphinx documentation engine to build outputs from your book's content.
- It uses a slightly modified version of the PyData Sphinx theme for beautiful HTML output.
- It uses a collection of Sphinx plugins and tools to add new functionality.

For more information about the project behind many of these tools, see The Executable Book Project documentation.

Contribute to Jupyter Book

Jupyter Book is an open project and we welcome your feedback and contributions! To contribute to Jupyter Book, see Contribute to Jupyter Book.

Acknowledgements

Jupyter Book is supported by an open community of contributors, many of whom come from the Jupyter community. Jupyter Book and many of the tools it uses are stewarded by the Executable Book Project, which is supported in part by the Alfred P. Sloan foundation.

PDFs for your book

It is possible to build a single PDF that contains all of your book's content. This page describes a couple ways to do so.

Warning

PDF building is experimental, and may change or have bugs.

There are two approaches to building PDF files.

Build a PDF from your book HTML

It is possible to build a single PDF from your book's HTML. This starts by converting all of your book's content into a single HTML file, and then renders it as a PDF by emulating a browser from the command-line.

Installation

Your system will need to use `pyppeteer` to parse the generated HTML for conversion to PDF.

You can install it like so:

```
pip install pyppeteer
```

You may also need to install this bundle of packages below (on *nix systems):

```
gconf-service
libasound2
libatk1.0-0
libatk-bridge2.0-0
libc6
libcairo2
libcups2
libdbus-1-3
libexpat1
libfontconfig1
libgcc1
libgconf-2-4
libgdk-pixbuf2.0-0
libglib2.0-0
libgtk-3-0
libnspr4
libpango-1.0-0
libpangocairo-1.0-0
libstdc++6
libx11-6
libx11-xcb1
libxcb1
libxcomposite1
libxcursor1
libxdamage1
libxext6
libxfixes3
libxi6
libxrandr2
libxrender1
libxss1
libxtst6
ca-certificates
fonts-liberation
libappindicator1
libnss3
lsb-release
xdg-utils
wget
```

In addition, if you get errors about libraries that don't exist, check out these install commands to see if that fixes it. We warned you it was an experimental feature :-)

Build

To build a single PDF from your book's HTML, use the following command:

```
jupyter-book build mybookname/ --builder pdfhtml
```

or

```
jb build mybookname/ --builder pdfhtml
```

Warning

If you get a "MaxRetryError" and see mentions of SSL in the error message when building the PDF, this could be due to a bug in `pyppeteer` as it downloads Chromium for the first time. See this GitHub comment for a potential fix, and this Jupyter Book issue where we're tracking the issue.

Control the look of PDF via HTML

Because you are using HTML as an intermediary for your book's PDF, you can control the look and feel of the HTML via your own CSS rules. Most CSS changes that you make to your HTML website will also persist in the PDF version of that website. For information about how to define your own CSS rules, see Custom CSS or JavaScript.

To add CSS rules that **only apply to the printed PDF**, use the `@media print` CSS pattern to define print-specific rules. These will *only* be applied when the HTML is being printed, and will not show up in your non-PDF website.

For example, to **hide the right table of contents** at print time, you could add this rule:

```
@media print {
    .bd-toc {
        visibility: hidden;
    }
}
```

The right Table of Contents would be present in your live website, but hidden when someone printed a PDF of your website.

Build a PDF using LaTeX

You can also use LaTeX to build a PDF of your book. This can behave differently depending on your operating system and `tex` setup. This section tries to recommend a few best-practices.

> **Note**
>
> We recommend using the texlive distribution

The default is to build your project as a single PDF file, however it is possible to build individual PDF files for each page of the project by enabling the `--individualpages` option when using the `pdflatex` builder.

Installation

For `Debian`-based `Linux` platforms it is recommended to install the following packages:

```
sudo apt-get install texlive-latex-recommended texlive-latex-extra \
                     texlive-fonts-recommended texlive-fonts-extra \
                     texlive-xetex latexmk
```

Alternatively you can install the full TeX Live distribution.

For `osx` you may want to use MacTeX which is a more user friendly approach. Alternatively you may also use TeX Live.

For `Windows` users, please install TeX Live.

Build

`jupyter-book` uses the jupyterbook-latex package which handles much of the customised LaTeX infrastructure. A feature list of this package can be found here. It enables building `pdf` files with full support for the `file` and `part/chapter` structures that are defined in the _toc.yml.

If you need to **turn off** this package, the following config is required:

```
latex:
  use_jupyterbook_latex: false
```

To build a PDF of your project using LaTeX, use the following command:

```
jupyter-book build mybookname/ --builder pdflatex
```

or

```
jb build mybookname/ --builder pdflatex
```

> **Note**
>
> If you would just like to generate the **latex** file you may use:
>
> ```
> jb build mybookname/ --builder latex
> ```

Individual PDF Files:

⚠ **Warning**

The current implementation of `--individualpages` does **not** make use of the improvements introduced by jupyterbook-latex and uses the default `latex` writer included with Sphinx. We are currently working on making improvements to how `--individualpages` are constructed. You can track progress here

To build PDF files for each page of the project, you can specify the option `--individualpages` for `--builder=pdflatex`.

The individual PDF files will be available in the `_build/latex` build folder. These files will have the same name as the source file or, if nested in folders, will be named `{folder}-{filename}.pdf`.

ℹ **Note**

When specifying a page using the `build` command, the `--individualpages` will automatically be set to `True`.

In the future we intend for this to produce latex documents more suitable to single pages (see issue #904).

Updating the name of the Global PDF file

To update the name of your `PDF` file you can set the following in `_config.yml`

```
latex:
  latex_documents:
     targetname: book.tex
```

This will act as an automatic `override` when Sphinx builds the latex_documents. It is typically inferred by `Sphinx` but when using `jupyter-book` naming the file in the `_config.yml` generally makes it easier to find.

Using a different LaTeX engine

The current default is to use `xelatex` to build `pdf` files.

⚠ **Warning**

The `--individualpages` option currently uses `pdflatex` by default.

Some users may want to switch to using a different LaTeX engine such as `pdflatex`. To revert the `LaTeX` engine to `pdflatex` you can add the following to your `_config.yml`

```
latex:
  latex_engine: pdflatex
```

ℹ **Note**

The Sphinx documentation for available builders contains a full list of supported `latex` builders.

Other Sphinx LaTeX settings

Other LaTeX settings available to Sphinx can be passed through using the config section of `Sphinx` in the `_config.yml` file for your project.

For example, if you would like to set the latex_toplevel_sectioning option to use `part` instead of `chapter` you would use:

```
sphinx:
  config:
    latex_toplevel_sectioning: 'part'
```

Custom Sphinx configuration

Jupyter Book uses the excellent documentation tool Sphinx to build your book and manage citations, cross-references, and extensibility.

While Jupyter Book comes pre-configured with several Sphinx extensions, power-users may wish to add their own extensions and configuration. This page describes how to do so.

Warning

Adding your own Sphinx configuration and extensions may cause Jupyter Book to behave unpredictably. Use at your own risk!

Custom Sphinx extensions

To enable your own Sphinx extensions when building a Jupyter Book, use the following configuration in your `_config.yml` file:

```
sphinx:
  extra_extensions:
   - extension1
   - extension2
```

Any extensions that are listed will be appended to the list of Sphinx extensions at build time.

Note

Make sure that you have your extension installed on your machine, or Sphinx won't know how to build the extensions.

An example: `sphinx-inline-tabs`

By default, Jupyter Book ships with tabs via `sphinx-panels`. There are other packages for tabs in the Sphinx ecosystem with different functionality. One-such package is `sphinx-inline-tabs`, which allows for *syncronized tabs* in case you'd like your tabs to shift across the page at the same time.

`sphinx-inline-tabs` is not included with Jupyter Book by default, but we can activate it with Jupyter Book like so:

- **Install `sphinx-inline-tabs`**. Here's the command to do so:

  ```
  pip install sphinx-inline-tabs
  ```

- **Add `sphinx-inline-tabs` content to your book**. Here's an example with MyST Markdown:

~~~
First two tabs showing off defining a function.

````{tab} Python
```python
def main():
    return
~~~
~~~

````{tab} C++
```c++
int main(const int argc, const char **argv) {
  return 0;
}
```
~~~

Second two tabs showing off printing.

````{tab} Python
```python
print("Hello World!")
```
````

````{tab} C++
```c++
#include <iostream>

int main() {
  std::cout << "Hello World!" << std::endl;
}
```
````
```

- **Activate `sphinx-inline-tabs` in `_config.yml`.** The `sphinx-inline-tabs` documentation says we activate it in Sphinx by adding `extensions = ["sphinx_inline_tabs"]`, so we'll add it to our Jupyter Book like so:

```
sphinx:
  extra_extensions:
  - sphinx_inline_tabs
```

Now, Jupyter Book will know how to interpret the `{tab}` directive (and any other directives that `sphinx-inline-tabs` supports).

For example, here is a rendered version of the tab code pasted above:

First two tabs showing off defining a function.

Python **C++**

```
def main():
    return
```

Second two tabs showing off printing.

Python **C++**

```
print("Hello World!")
```

Local Sphinx Extensions

Sphinx is able to use local extensions by adding additional directories to the Python path. You can use local extensions by specifying them as local_extensions in the `_config.yml` file.

Custom CSS or JavaScript

If you'd like to include custom CSS rules or JavaScript scripts in your book, you can do so by adding them to a folder called `_static` in your book's folder. Any files that end in `.css` or `.js` in this folder will automatically be copied into your built book HTML and linked in the header of each page.
```

For example, to include a custom CSS file `myfile.css` in a Jupyter Book folder with the following structure:

```
mybook/
├── _config.yml
├── _toc.yml
└── page1.md
```

Add the static file here:

```
├── _config.yml
├── _toc.yml
├── page1.md
└── _static
    └── myfile.css
```

The rules should then automatically be applied to your site. In general, these CSS and JS files will be loaded *after* others are loaded on your page, so they should overwrite pre-existing rules and behaviour.

An example: justify the text

If you want the text of you book to be justified instead of left aligned then create `myfile.css` under `mybook/_static` with the following CSS:

```css
p {
    text-align: justify;
}
```

Manual Sphinx configuration

You may also directly override the key-value pairs that Sphinx normally has you configure in `conf.py`. To do so, use the following section of `_config.yml`:

```yaml
sphinx:
  config:
    key1: value1
    key2: value2
```

Warning

Any options set in this section will **override** default configurations set by Jupyter Book. Use at your own risk!

Tip

If you wish to inspect a `conf.py` representation of the generated configuration, which Jupyter Book will pass to Sphinx, you can run from the command-line:

```
jb config sphinx mybookname/
```

Fine control of parsing and execution

As discussed in the components of Jupyter Book, two of the main components of Jupyter Book are Sphinx extensions; MyST-Parser for Markdown parsing, and MyST-NB for notebook execution and output rendering.

These two extensions are highly customizable *via* Sphinx configuration. Some of their configuration is already exposed in the `_config.yml`, but you can also directly set configuration, see:

- the MyST-Parser configuration options
- the MyST-NB configuration options

Defining TeX macros

You can add LaTeX macros for the whole book by defining them under the `Macros` option of the `TeX` block. For example, the following two macros have been pre-defined in the Sphinx configuration

```yaml
sphinx:
  config:
    mathjax_config:
      TeX:
        Macros:
          "N": "\\mathbb{N}"
          "floor": ["\\lfloor#1\\rfloor", 1]
          "bmat" : ["\\left[\\begin{array}"]
          "emat" : ["\\end{array}\\right]"]
```

You can also define TeX macros for a specific file by introducing them at the beginning of the file under a `math` directive. For example

~~~
```{math}

\newcommand\N{\mathbb{N}}
\newcommand\floor[1]{\lfloor#1\rfloor}
\newcommand{\bmat}{\left[\begin{array}}
\newcommand{\emat}{\end{array}\right]}
```
~~~

The commands can be used inside a `math` directive, `$$`, or in-line `$`. For example,

```
$$
A = \bmat{} 1 & 1 \\ 2 & 1\\ 3 & 2 \emat{},\ b=\bmat{} 2\\ 3 \\ 4\emat{},\ \gamma = 0.5
$$
```

will be rendered as:

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 2 \end{bmatrix},\ b = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix},\ \gamma = 0.5$$

See also

How MyST-Parser works with MathJax, and the section on math and equations.

Important

To have “bare” LaTeX rendered in HTML, enable the `amsmath` extension in your `_config.yml`:

```yaml
parse:
  myst_enable_extensions:
    # don't forget to list any other extensions you want enabled,
    # including those that are enabled by default!
    - amsmath
```

Then you can include:

```latex
\begin{equation}
  \int_0^\infty \frac{x^3}{e^x-1}\,dx = \frac{\pi^4}{15}
\end{equation}
```

which renders as

$$\int_0^\infty \frac{x^3}{e^x - 1}\,dx = \frac{\pi^4}{15} \tag{5}$$

How-to and FAQ

This page contains more advanced and complete information about the `jupyter-book` repository. See the sections below.

Enable Google Analytics

If you have a Google account, you can use Google Analytics to collect some information on the traffic to your Jupyter Book. With this tool, you can find out how many people are using your book, where they come from and how they access it, whether they are using the desktop or the mobile version etc.

To add Google Analytics to your Jupyter Book, navigate to Google Analytics, create a new Google Analytics account and add the url of your Jupyter Book to a new *property*. Once you have set everything up, your Google Analytics property will have a so-called Tracking-ID, that typically starts with the letters UA. All that you need to do is to copy this ID and paste it into your configuration file:

```
html:
  google_analytics_id: UA-XXXXXXXXX-X
```

Check external links in your book

If you'd like to make sure that the links outside of your book are valid, run the Sphinx link checker with Jupyter Book. This will check each of your external links and ensure that they resolve.

To run the link checker, use the following command:

Note that you must ensure each link is the *right* target, the link checker will only ensure that it resolves.

```
jupyter-book build mybookname/ --builder linkcheck
```

It will print the status of each link in your book so that you may resolve any incorrect links later on.

Clean your book's generated files

It is possible to "clean up" the files that you generate when you build your book. This is often useful if you have recently changed a lot of content in order to ensure that you build your book from a clean slate.

You can clean up your book's generated content by running the following command:

```
jupyter-book clean mybookname/
```

By default, this will delete all folders inside `mybookname/_build` *except* for a folder called `.jupyter_cache`. This ensures that the *content* of your book will be regenerated, while the cache that is generated by *running your book's code* will not be deleted (because regenerating it may take some time).

To delete the `.jupyter_cache` folder as well, add the `--all` flag like so:

```
jupyter-book clean mybookname/ --all
```

This will entirely remove the folders in the `_build/` directory.

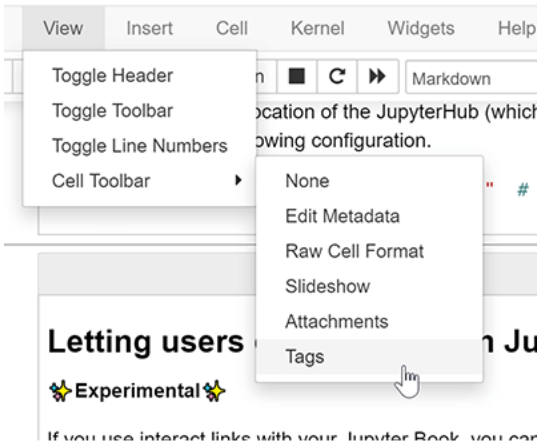
How should I add cell tags and metadata to my notebooks?

You can control the behaviour of Jupyter Book by putting custom tags in the metadata of your cells. This allows you to do things like automatically hide code cells as well as add interactive widgets to cells.

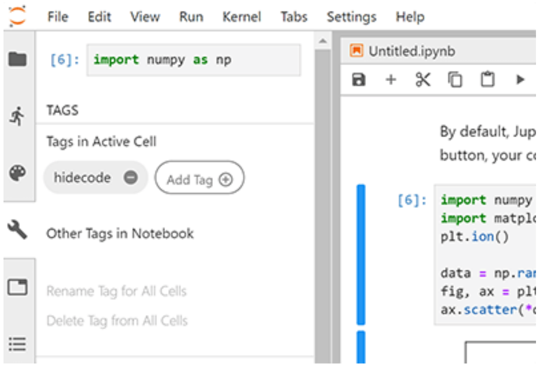
Adding tags using notebook interfaces

There are two straightforward ways to add metadata to cells:

1. **Use the Jupyter Notebook cell tag editor**. The Jupyter Notebook ships with a cell tag editor by default. This lets you add cell tags to each cell quickly.
   To enable the cell tag editor, click `View -> Cell Toolbar -> Tags`. This will enable the tags UI. Here's what the menu looks like.



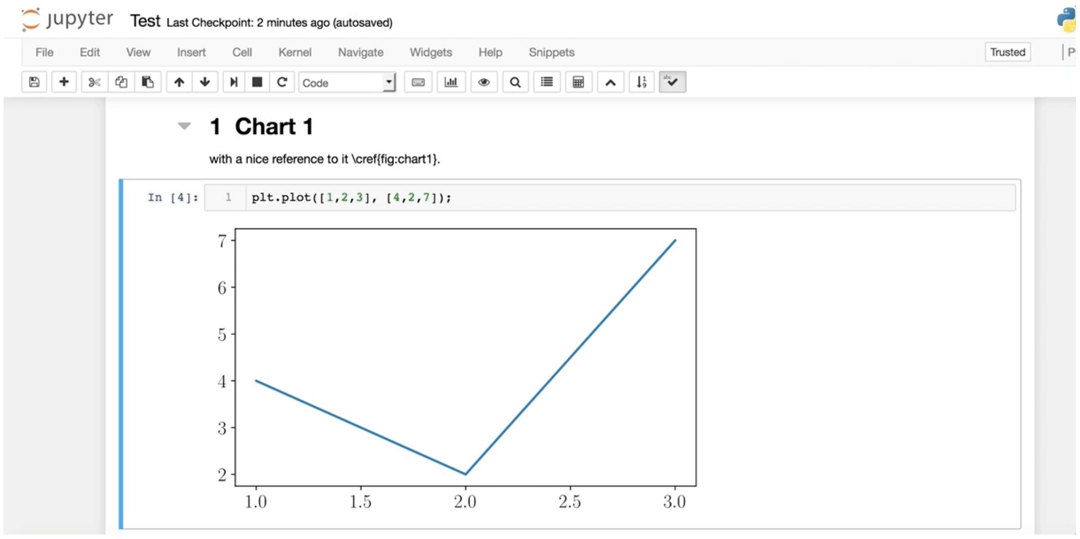
2. **Use the JupyterLab Cell Tags plugin**. JupyterLab is an IDE-like Jupyter environment that runs in your browser. It has a "cell tags" plugin built-in, which exposes a user interface that lets you quickly insert cell tags. You'll find tags under the "wrench" menu section. Here's what the tags UI in JupyterLab looks like.



```
[6]: import numpy as np
```

Tags are actually just a special section of cell level metadata. There are three levels of metadata:

- For notebook level: in the Jupyter Notebook Toolbar go to `Edit -> Edit Notebook Metadata`
- For cell level: in the Jupyter Notebook Toolbar go to `View -> Cell Toolbar -> Edit Metadata` and a button will appear above each cell.
- For output level: using e.g. `IPython.display.display(obj,metadata={"tags": []})`, you can set metadata specific to a certain output (but Jupyter Book does not utilize this just yet).



```
In [4]: plt.plot([1,2,3], [4,2,7]);
```

Add tags using MyST Markdown notebooks

If you're writing notebooks with MyST Markdown, then you can add tags to each code cell when you write the `{code-cell}` block. For example, below we:

~~~
```{code-cell}
:tags: [tag1,tag2,tag3]
print("some code")
```
~~~

Will create a code cell with those three tags attached to it. For more information about MyST Markdown notebooks, see Notebooks written entirely in Markdown.

Add tags using Python code

Sometimes you'd like to quickly scan through a notebook's cells in order to add tags based on the content of the cell. For example, you might want to hide any cell with an import statement in it using the `remove-input` tag.

Here's a short Python snippet to accomplish something close to this. First change directories into the root of your book folder, and then run the script below as a Python script or within a Jupyter Notebook (modifying as necessary for your use case). Finally, check the changes that will be made and commit them to your repository.

```python
import nbformat as nbf
from glob import glob

# Collect a list of all notebooks in the content folder
notebooks = glob("./content/**/*.ipynb", recursive=True)

# Text to look for in adding tags
text_search_dict = {
    "# HIDDEN": "remove-cell",  # Remove the whole cell
    "# NO CODE": "remove-input",  # Remove only the input
    "# HIDE CODE": "hide-input"  # Hide the input w/ a button to show
}

# Search through each notebook and look for the text, add a tag if necessary
for ipath in notebooks:
    ntbk = nbf.read(ipath, nbf.NO_CONVERT)

    for cell in ntbk.cells:
        cell_tags = cell.get('metadata', {}).get('tags', [])
        for key, val in text_search_dict.items():
            if key in cell['source']:
                if val not in cell_tags:
                    cell_tags.append(val)
        if len(cell_tags) > 0:
            cell['metadata']['tags'] = cell_tags

    nbf.write(ntbk, ipath)
```

Use raw HTML in Markdown

Jupyter Notebook Markdown allows you to use raw HTML in Markdown cells. This is discouraged in most cases, because it will usually just be passed through the build process as raw text, and so will not be subject to processes like:

- relative path corrections
- copying of assets to the build folder
- multiple output type formatting (e.g. it will not show in PDFs!).

So, for instance, below we add:

```
<a href="../intro.md">Go Home HTML!</a>

[Go Home Markdown!](../intro.md)
```

and you will find that the HTML link is broken:

Go Home HTML!

Go Home Markdown!

Tip

Note that MyST Markdown now has some extended syntax features, which can allow you to use certain HTML elements in the correct manner.

For example, the raw HTML image tag

```
<img src="../images/fun-fish.png" alt="the fun fish!" width="200px"/>
```

becomes

See the image section for details.

Adding extra HTML to your book

There are a few places in Jupyter Book where you can add extra arbitrary HTML. In all cases, this is done with a configuration value in your `_config.yml` file.

Extra HTML in your footer

To add extra HTML in your book's footer, use the following configuration:

```
html:
    extra_footer: |
        <div>
            your html
        </div>
```

The contents of `extra_footer` will be inserted into your page's HTML after all other footer content.

Extra HTML to your left navbar

To add extra HTML in your book's left navbar, use the following configuration:

```
html:
    extra_navbar: |
        <div>
            your html
        </div>
```

The contents of `extra_navbar` will be inserted into your page's HTML after all other HTML content.

Adding a license to your HTML footer

If you'd like to add a more detailed license for your book, or would like to add a link to an external page for a license, the easiest way to do so is to use a custom footer for this. You can disable the "copyright" text that is automatically added to each footer, and add whatever footer HTML you'd like.

For example, see this configuration:

```
html:
  extra_footer: |
    <p>
    ... Add license info here...
    </p>
sphinx:
  config:
    html_show_copyright: false
```

Note that this may not work in PDF builds of your page generated by LaTeX.

Configuring to Improve Accessibility

Declaring the primary language used in your book assists screen reader and browser translation tools.

Language can be configured by providing the appropriate language code to the `language` option, under `sphinx` configuration in your `_config.yml` file:

```yaml
sphinx:
  config:
    language: en
```

This example will set the book language to English, which would be represented in your book's HTML as `<html lang="en">...</html>`.

Working on Windows

Jupyter Book is now also tested against a Windows environment on Python 3.7 😀

For its specification, see the `windows-latest` runner used by GitHub CI.

However, there is a known incompatibility for notebook execution, when using Python 3.8 (see issue #906).

If you're running a recent version of Windows 10 and encounter any issues, you may also wish to try installing Windows Subsystem for Linux.

As of June 5, 2020, there were three open issues that required Windows-specific changes. We hope these are now fixed in version 0.8 of Jupyter Book but, in case any issues still arise, we leave these community tips, which are known to work for some users. Note that there is no guarantee that they will work on all Windows installations.

1. Character encoding
   Jupyter Book currently reads and writes files on Windows in the native Windows encoding, which causes encoding errors for some characters in UTF8 encoded notebooks.
   Work-around: Beginning with Python 3.7 cmd.exe or powershell enviroments that set PYTHONUTF8=1 override the native locale encoding and use UTF8 for all input/output.

   > **Tip**
   >
   > To make it easier to use this option, the EOAS/UBC notebook courseware project has created a Conda package runjb which does this automatically for powershell

2. A new Windows event loop
   The asyncio event loop has been changed for Python 3.8 causing sphinx-build to fail.
   Work-around: Pin to Python 3.7.6. This environment_win.yml file does that, and also installs runjb to fix issue 1.
3. Nested tables of contents
   Currently, `_toc.yml` files that reference Markdown files in sub-folders are failing for some Windows users. That is, this original _toc.yml file will fail with a message saying Jupyter Book "`cannot find index.md`"
   Work-around: Flatten the layout of the book to a single level, i.e. this _toc.yml file works with Windows.

Summary

The following workflow should succeed using a miniconda powershell terminal on Windows 10:

1. `conda install git`
2. `git clone https://github.com/eoas-ubc/quantecon-mini-example.git`
3. `cd quantecon-mini-example`
4. `git checkout windows`
5. `conda env create -f environment_win.yml`
6. `conda activate wintest`
7. `cd mini_book`
8. `runjb docs`

After the build, view the HTML with:

```
start docs\_build\html\index.html
```

Manually specify extra files/folders to be included in a website

Jupyter Book will copy over any files that are linked from within its pages so that the links work in the built website. However, sometimes you'd like to manually ensure that files and folders are included in your built website. For example, if you'd like to link to them from *outside* your built documentation, but not from within your built documentation.

To manually specify items to copy over, use the `html_extra_path` Sphinx configuration. You can configure this with Jupyter Book like so:

```
sphinx:
  config:
    html_extra_path: ['folder1', 'folder2']
```

When you build your book's HTML, Jupyter Book will ensure that all files and folders *inside* the folders specified in `html_extra_path` will be copied over to your built website.

For example, if you have a folder structure in your book like so:

```
assets
└── data
    └── mydataset.csv
```

and the following Jupyter Book configuration:

```
sphinx:
  config:
    html_extra_path: ['assets']
```

Then the dataset will be accessible at `yourwebsite.com/data/mydataset.csv`.

Enabling a custom builder using `jupyter-book`

You can initiate builds for a custom builder using:

```
jb build <project> --builder=custom --custom-builder=<builder-name>
```

Advanced `sphinx` users may find an extension that builds a different type of output from the Sphinx AST such as sphinx-tojupyter which is an extension for building notebooks that only includes `basic` markdown.

> ⚠ **Warning**
>
> sphinx-tojupyter will be deprecated once `myst` syntax rendering support is available in jupyter notebooks.

You can enable the `jupyter` builder by adding it to the `_config.yml`

```
sphinx:
  extra_extensions: [sphinx_tojupyter]
```

and using the `custom` option via `jupyter-book`:

```
jb build <project> --builder=custom --custom-builder=jupyter
```

> ⚠ **Warning**
>
> **Developers:** When using other output targets, the package will need to support specifying the `mime` type priority for `myst_nb` compatibility.
>
> See this code for further details

What if I have an issue or question?

If you've got questions, concerns, or suggestions, please open an issue at at the Jupyter Book issues page

Contribute to Jupyter Book

Welcome to the `jupyter-book` repository! We're excited you're here and want to contribute. ✨

Development guidelines

For information about development conventions, practices, and infrastructure, please see the `executablebooks/` development guidelines.

Getting started

To get started with Jupyter Book's *codebase*, take the following steps:

Clone and install the package

```
git clone https://github.com/executablebooks/jupyter-book
cd jupyter-book
```

Next, install:

```
pip install -e .[all]
```

This will install Jupyter Book locally, along with the packages needed to test it as well as packages for ensuring code style.

Install the pre-commit hooks

Jupyter Book uses pre-commit to ensure code style and quality before a commit is made. This ensures that the look and feel of Jupyter Book remains consistent over time and across developers. `pre-commit` is installed when you install Jupyter Book with `pip install -e .[code_style]`.

To enable `pre-commit` for your clone, run the following from the repository root:

```
pre-commit install
```

From now on, when you make a commit to Jupyter Book, `pre-commit` will ensure that your code looks correct according to a few checks.

Run the tests

For code tests, Jupyter Book uses pytest). You may run all the tests, or only ones that do not require additional installations, with the following command:

```
>> pytest -m 'not requires_chrome and not requires_tex'
```

You can alternatively use tox to run the tests in multiple isolated environments, and also without the need for the initial dependencies install (see the `tox.ini` file for available test environments and further explanation):

```
>> tox -e py38-sphinx3 -- -m 'not requires_chrome and not requires_tex'
```

Either will run the Jupyter Book test suite, *except for the PDF tests*. This is because running the PDF generation tests requires a full LaTeX environment, which you may not have set up.

Note

Jupyter Book makes use of pytest-xdist for running tests in parallel. You can take advantage of this by running tests with the `-n` argument followed by the number of CPUs you would like to use. For example: `pytest -n 4`. This makes the tests run much faster.

To test PDF generation

If you'd like to test (or try out) the generation of PDFs, take the following steps:

To generate PDFs via HTML, make sure you install Jupyter Book with `pip install -e .[pdfhtml]`. This will install `pyppeteer`, which runs a headless chrome session to convert your book to PDF. Next, follow the installation instructions at Build a PDF from your book HTML. You should then be able to build your book's PDF through HTML.

To generate PDFs via LaTeX, make sure you install a working LaTeX distribution locally. Do so by following the instructions in Build a PDF using LaTeX.

If you have installed the requirements for both HTML and LaTeX generation, you should be able to run the full test suite with pytest.

GitHub Actions Artifacts

A test included for each pull request is to build the `docs` as `PDF` files using both the `pdfhtml` and `pdflatex` writers. These tests build the `pdf` file and then save them as artifacts attached to each workflow run.

These `pdf` files can be retrieved from the top right corner of a workflow run.

Repository structure of Jupyter Book

This section covers the general structure of the Jupyter Book repository, and explains which pieces are where.

The Jupyter Book repository contains two main pieces:

MyST Markdown

Jupyter Book supports an *extended version of Jupyter Markdown* called "MyST Markdown". For information about the MyST syntax and how to use it, see the MyST-Parser documentation.

The command-line tool and Python package

This is used to help create and build books. It can be found at `./jupyter_book`.

- **The `commands/` folder has the CLI**. This is the interface for users to create, build and control their book via the command-line.
- **The `sphinx.py` module builds the books**.
- **The `yaml.py` module handles configuration**.
- **The `toc.py` module prepares the table of contents**.

The other modules handle more specific functionality in Jupyter Book - see their module docstrings for more information.

The template Jupyter Book

Jupyter Book comes bundled with a small template book to show off content. This can be immediately built with `jupyter-book build`. It can be found at `jupyter_book/book_template`.

An example

Here are a few examples of how this code gets used to help you get started.

- when somebody runs `jupyter-book create mybook/`, the `create.py` module is used to generate an empty template using the template in `jupyter_book/book_template/`.

- when somebody runs `jupyter-book build mybook/`, the `build.py` module loops through your page content files, and uses the `page/` module to convert each one into HTML and places it in `mybook/_build`.

Hopefully this explanation gets you situated and helps you understand how the pieces all fit together. If you have any questions, feel free to open an issue asking for help!

Other major tools in the Jupyter Book stack

Jupyter Book depends on a collection of open source tools in the Python / Sphinx ecosystem. These tools do much of the heavy lifting of Jupyter Book, and are where many improvements and changes will need to be. Here is a list of the major tools and what kinds of functionality they support:

- The Sphinx Documentation engine is used to build book outputs. This relies on a number of extensions that are developed by Jupyter Book.
- MyST Markdown is parsed into Sphinx by the MyST-Parser.
- The MyST-NB package parses Jupyter Notebooks into Sphinx and also controls some parts of notebook execution. It also allows inserting code outputs into content.
- Jupyter-Cache manages the execution and cacheing of notebook content at build time. It is controlled by MyST-NB.
- The Sphinx-Book-Theme defines the look and feel of Jupyter Book, and is where most of the CSS rules are defined.

MyST cheat sheet

Headers

Syntax	Example	Note

`# Heading level 1`
`## Heading level 2`
`### Heading level 3`
`#### Heading level 4`
`##### Heading level 5`
`###### Heading level 6`
	`# MyST Cheat Sheet`	Level 1-6 headings, denoted by number of `#`

Target headers

Syntax	Example	Note

`(target_header)=`	`(myst_cheatsheet)=`
	`# MyST Cheat Sheet`
		See below how to reference target headers.

Referencing target headers

Targets can be referenced with the ref inline role which uses the section title by default:

```
{ref}`myst_cheatsheet`
```

You can specify the text of the target:

```
{ref}`MyST syntax lecture <myst_cheatsheet>`
```

Another alternative is to use Markdown syntax:

```
[MyST syntax lecture](myst_cheatsheet)
```

Quote

Syntax	Example	Note
`> text`	`> this is a quote`	quoted text

Thematic break

Syntax	Example	Note
`---`	`This is the end of some text.` `---` `## New Header`	Creates a horizontal line in the output

Line comment

Syntax	Example	Note
`% text`	`a line` `% a comment` `another line`	See Comments for more information.

Block break

Syntax	Example	Result
`+++`	`This is an example of` `+++ {"meta": "data"}` `a block break`	This is an example of a block break

HTML block

Syntax	Example	Result
`<tagName> text <tagName>`	`<p> This is a paragraph </p>`	This is a paragraph

Links

Syntax	Example	Result
`[text](target)`	`[Jupyter Book](https://jupyterbook.org)`	Jupyter Book
`[text](relative_path)`	`[PDF documentation](../advanced/pdf)`	PDF documentation
`[text](target "title")`	`[Jupyter Book](https://jupyterbook.org "JB Homepage")`	Jupyter Book
`<target>`	`<https://jupyterbook.org>`	https://jupyterbook.org
`[text][key]`	`[Jupyter Book][intro_page]` `[intro_page]: https://jupyterbook.org`	Jupyter Book

Lists

Ordered list

Example	Result
`1. First item` `2. Second item` `    1. First sub-item`	1. First item 2. Second item 1. First sub-item
`1. First item` `2. Second item` `    * First sub-item`	1. First item 2. Second item ◦ First subitem

Unordered list

Example	Result
`* First item` `* Second item` `  * First subitem`	• First item • Second item ◦ First subitem
`* First item` `  1. First subitem` `  2. Second subitem`	• First item 1. First subitem 2. Second subitem

Tables

Syntax	Example	Result

`| a    | b    |`
`| :--- | ---: |`
`| c    | d    |`
	`|    Training    |    Validation`
	`|`
	`| :------------ | ------------`
	`-: |`
	`|       0       |        5`
	`|`
	`|     13720     |       2744`
	`|`
		Training Validation
		0 5
		13720 2744

`
```
{list-table}`
`:header-rows: 1`

`* - Col1`
`  - Col2`
`* - Row1 under Col1`
`  - Row1 under Col2`
`* - Row2 under Col1`
`  - Row2 under Col2`
`
```
 `
	`
	```
	{list-table}`
	`:header-rows: 1`
	`:name: example-table`

	`* - Training`
	`  - Validation`
	`* - 0`
	`  - 5`
	`* - 13720`
	`  - 2744`
	`
	```
	 `
		Training Validation
		0 5
		13720 2744
		
		Table 2 My table title

`
```
{list-table} Table title`
`:header-rows: 1`

`* - Col1`
`  - Col2`
`* - Row1 under Col1`
`  - Row1 under Col2`
`* - Row2 under Col1`
`  - Row2 under Col2`
`
```
 `
	`
	```
	{list-table} This table`
	`title`
	`:header-rows: 1`

	`* - Training`
	`  - Validation`
	`* - 0`
	`  - 5`
	`* - 13720`
	`  - 2744`
	`
	```
	 `
		Training Validation
		0 5
		13720 2744
		
		Table 3 This table title

Referencing tables

Note

In order to reference a table you must add a label to it. To add a label to your table simply include a `:name:` parameter followed by the label of your table. In order to add a numbered reference, you must also include a table title. See example above.

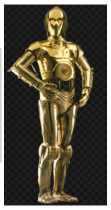
Syntax	Example	Result
`` {numref}`label` ``	`` {numref}`example-table` is an example. ``	Table 2 is an example.
`` {ref}`text <label>` ``	`` This {ref}`table <example-table>` is an example. ``	This table is an example.
`` {numref}`text %s <label>` ``	`` {numref}`Tbl %s <example-table>` is an example. ``	Tbl 2 is an example.

Admonitions

Syntax	Example	Result
`` ```{admonition} Title text ``` ``	`` ```{admonition} This is a title An example of an admonition with a title. ``` ``	**This is a title** An example of an admonition with a title.
`` ```{note} text ``` `` or `` ```{note} text some more text... ``` ``	`` ```{note} Notes require **no** arguments, so content can start here. ``` ``	**Note** Notes require **no** arguments, so content can start here.
`` ```{warning} text some more text... ``` ``	`` ```{warning} This is an example of a warning directive. ``` ``	**Warning** This is an example of a warning directive.
`` ```{tip} text some more text... ``` ``	`` ```{tip} This is an example of a tip directive. ``` ``	**Tip** This is an example of a tip directive.
`` ```{caution} text some more text... ``` ``	`` ```{caution} This is an example of a caution directive. ``` ``	**Caution** This is an example of a caution directive.
`` ```{attention} text some more text... ``` ``	`` ```{attention} This is an example of an attention directive. ``` ``	**Attention** This is an example of an attention directive.
`` ```{danger} text some more text... ``` ``	`` ```{danger} This is an example of a danger directive. ``` ``	**Danger** This is an example of a danger directive.
`` ```{error} text some more text... ``` ``	`` ```{error} This is an example of an error directive. ``` ``	**Error** This is an example of an error directive.
`` ```{hint} text some more text... ``` ``	`` ```{hint} This is an example of a hint directive. ``` ``	**Hint** This is an example of a hint directive.

Syntax	Example	Result
` ```{important} text some more text... ``` `	` ```{important} This is an example of an important directive. ``` `	**Important** This is an example of an important directive.

Figures and images

Syntax	Example	Result
` ```{figure} ./path/to/figure.jpg :name: label caption ``` `	` ```{figure} ../images/C-3PO_droid.png :height: 150px :name: figure-example Here is my figure caption! ``` `	**Fig. 12** Here is my figure caption!
` ```{image} ./path/to/figure.jpg :name: label ``` `	` ```{image} ../images/C-3PO_droid.png :height: 150px :name: image-example ``` `	

Note

Content is not permitted in the image directive.

See Images and figures and Markdown files for more information.

Referencing figures

Syntax	Example	Result
`` {numref}`label` ``	`` {numref}`figure-example`is a figure example. ``	Fig. 12 is a figure example.
`` {numref}`text %s <label>` ``	`` {numref}`Figure %s <figure-example>` is an example. ``	Figure 12 is an example.
`` {ref}`text <label>` ``	`` This {ref}`figure <figure-example>` is an example. ``	This figure is an example.

Referencing images

Syntax	Example	Result
`` {ref}`text <label>` ``	`` This {ref}`image <image-example>` is an example. ``	This image is an example.

Math

Syntax	Example	Result
Inline	`This is an example of an inline equation $z=\sqrt{x^2+y^2}$.`	This is an example of an inline equation $z=\sqrt{x^2+y^2}$.
Math blocks	`This is an example of a math block $$ z=\sqrt{x^2+y^2} $$`	This is an example of a math block $$z=\sqrt{x^2+y^2}$$
Math blocks with labels	`This is an example of a math block with a label $$ z=\sqrt{x^2+y^2} $$ (mylabel)`	This is an example of a math block with a label $$z=\sqrt{x^2+y^2} \qquad (6)$$
Math directives	```` This is an example of a math directive with a label ```{math} :label: eq-label z=\sqrt{x^2+y^2} ``` ````	This is an example of a math directive with a label $$z=\sqrt{x^2+y^2} \qquad (7)$$

See Math and equations for more information.

Referencing math directives

Syntax	Example	Result
``{eq}`label` ``	``Check out equation {eq}`eq-label`.``	Check out equation (7).

Code

In-line code

Example:

```
Wrap in-line code blocks in backticks: `boolean example = true;`.
```

Result:

Wrap in-line code blocks in backticks: `boolean example = true;`.

Code and syntax highlighting

Example:

~~~
```python
note = "Python syntax highlighting"
print(node)
```
~~~

or

```
```
No syntax highlighting if no language
is indicated.
```
```

Result:

```
note = "Python syntax highlighting"
print(node)
```

or

```
No syntax highlighting if no language
is indicated.
```

Executable code

⚠ **Warning**

Make sure to include this front-matter YAML block at the beginning of your `.ipynb` or `.md` files.

```
---
jupytext:
  formats: md:myst
  text_representation:
    extension: .md
    format_name: myst
kernelspec:
  display_name: Python 3
  language: python
  name: python3
---
```

Example:

~~~
```{code-cell} ipython3
note = "Python syntax highlighting"
print(note)
```
~~~

Result:

```
note = "Python syntax highlighting"
print(note)
```

```
Python syntax highlighting
```

See Notebooks written entirely in Markdown for more information.

Tags

The following `tags` can be applied to code cells by introducing them as options:

Tag option	Description	Example

`"full-width"`	Cell takes up all of the horizontal space	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["full-width"]` 
		 `print("This is a test.")` 
		 ``
		```
		 ``

`"output_scroll"`	Make output cell scrollable	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["output_scroll"]` 
		 `for ii in range(100):` 
		 `  print("This is a test.")` 
		 ``
		```
		 ``

`"margin"`	Move code cell to the right margin	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["margin"]` 
		 `print("This is a test.")` 
		 ``
		```
		 ``

`"hide-input"`	Hide cell but the display the outputs	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["hide-input"]` 
		 `print("This is a test.")` 
		 ``
		```
		 ``

`"hide-output"`	Hide the outputs of a cell	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["hide-output"]` 
		 `print("This is a test.")` 
		 ``
		```
		 ``

`"hide-cell"`	Hides inputs and outputs of code cell	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["hide-cell"]` 
		 `print("This is a test.")` 
		 ``
		```
		 ``

`"remove-input"`	Remove the inputs of a cell	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["remove-input"]` 
		 `print("This is a test.")` 
		 ``
		```
		 ``

`"remove-output"`	Remove the outputs of a cell	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["remove-output"]` 
		 `print("This is a test.")` 
		 ``
		```
		 ``

`"remove-cell"`	Remove the entire code cell	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["remove-cell"]` 
		 `print("This is a test.")` 
		 ``
		```
		 ``

`"raises-exception"`	Mark cell as “expected to error”	``
		```
		{code-cell} ipython3 `` 
		 `:tags: ["raises-exception"]` 
		 `while True print('Hello world')` 
		 ``
		```
		 ``

Gluing variables

Example:

~~~
```{code-cell} ipython3
from myst_nb import glue
my_variable = "here is some text!"
glue("glued_text", my_variable)
```

Here is an example of how to glue text: {glue:}`glued_text`
~~~

Result:

```python
from myst_nb import glue
my_variable = "here is some text!"
glue("glued_text", my_variable)
```

```
'here is some text!'
```

Here is an example of how to glue text: `'here is some text!'`

See Gluing variables in your notebook for more information.

Gluing numbers

Example:

~~~
```{code-cell} ipython3
from myst_nb import glue
import numpy as np
import pandas as pd

ss = pd.Series(np.random.randn(4))
ns = pd.Series(np.random.randn(100))

glue("ss_mean", ss.mean())
glue("ns_mean", ns.mean(), display=False)
```

Here is an example of how to glue numbers: {glue:}`ss_mean` and {glue:}`ns_mean`.
~~~

Result:

```python
from myst_nb import glue
import numpy as np
import pandas as pd

ss = pd.Series(np.random.randn(4))
ns = pd.Series(np.random.randn(100))

glue("ss_mean", ss.mean())
glue("ns_mean", ns.mean(), display=False)
```

```
-0.28253689832716616
```

Here is an example of how to glue numbers: `-0.28253689832716616` and `-0.004945890972819941`.

See Gluing variables in your notebook for more information.

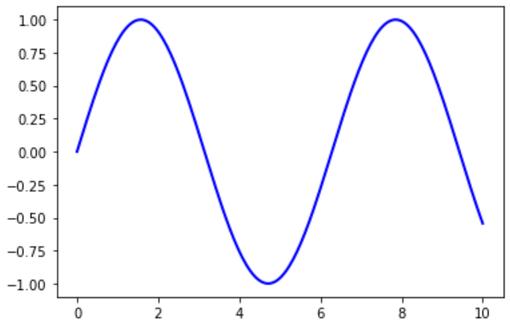
Gluing visualizations

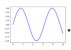
Example:

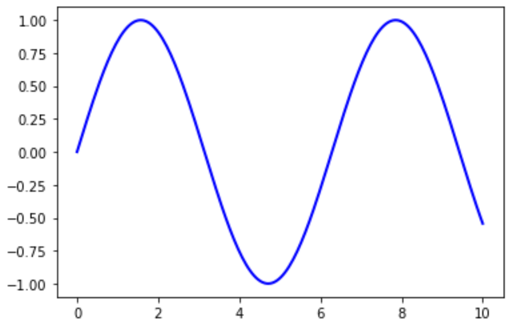
~~~
```{code-cell} ipython3
from myst_nb import glue
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 200)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y, 'b-', linewidth=2)

glue("glued_fig", fig, display=False)
```

This is an inline glue example of a figure: {glue:}`glued_fig`.
This is an example of pasting a glued output as a block:
```{glue:} glued_fig
```
~~~

Result:

```python
from myst_nb import glue
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 200)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y, 'b-', linewidth=2)

glue("glued_fig", fig, display=False)
```



This is an inline glue example of a figure: . This is an example of pasting a glued output as a block:



See Gluing variables in your notebook for more information.

Gluing math

Example:

~~~
```{code-cell} ipython3
import sympy as sym
x, y = sym.symbols('x y')
z = sym.Function('z')
z = sym.sqrt(x**2+y**2)
glue("example_eq", z, display=False)
```

To glue a math equation try
```{glue:math} example_eq
:label: glue-eq-example
```
~~~

Result:

```python
import sympy as sym
x, y = sym.symbols('x y')
z = sym.Function('z')
z = sym.sqrt(x**2+y**2)
glue("example_eq", z, display=False)
```

To glue a math equation try:

$$\sqrt{x^2 + y^2} \qquad ()$$

See Insert code outputs into page content for more information.

Reference documents

Syntax	Example	Result
``{doc}`path/to/document` ``	``See {doc}`../content/citations` for more information.``	See References and citations for more information.
``{doc}`text <path/to/document>` ``	``See {doc}`here <../content/citations>` for more information.``	See here for more information.

Footnotes

Syntax	Example	Result

`[^ref]`
 	
`[^ref]: Footnote text`
	`This is an example of a footnote. [^footnote1]`
	
	`[^footnote1]: The definition for referencing footnotes is generally placed at the bottom of the document.`
		This is a footnote reference.[1]

Note

Footnotes are displayed at the very bottom of the page.

See Footnotes for more information.

Citations

Note

Make sure you have a reference bibtex file. You can create one by running `touch references.bib` or view a `references.bib` example.

Syntax	Example	Result
``{cite}`mybibtexcitation` ``	``This example generates the following citation {cite}`perez2011python`.``	This example generates the following citation [PGH11].

To include a list of citations mentioned in the document, introduce the `bibliography` directive

~~~
```{bibliography}
:filter: docname in docnames
```
~~~

See References and citations for more information.

[1] This **is** the footnote definition.

The command-line interface

Jupyter Book comes with a command-line interface that makes it easy to build your books and run a few common functions. This page contains information on what you can do with the CLI.

This page is a complete reference for the CLI. For newcomers who would like to get started with the Jupyter Book CLI, we recommend starting with Overview

Note

You may also use `jb` as shorthand for `jupyter-book` in the command-line. For example: `jupyter-book build mybook/` is equivalent to `jb build mybook/`.

See below for the full command-line reference

jupyter-book

Build and manage books with Jupyter.

```
jupyter-book [OPTIONS] COMMAND [ARGS]...
```

Options

`--version`
Show the version and exit.

build

Convert your book's or page's content to HTML or a PDF.

```
jupyter-book build [OPTIONS] PATH_SOURCE
```

Options

`--path-output <path_output>`
Path to the output artifacts

`--config <config>`
Path to the YAML configuration file (default: PATH_SOURCE/_config.yml)

`--toc <toc>`
Path to the Table of Contents YAML file (default: PATH_SOURCE/_toc.yml)

`-W, --warningiserror`
Error on warnings.

`-n, --nitpick`
Run in nit-picky mode, to generates warnings for all missing references.

`--keep-going`
With -W, do not stop the build on the first warning, instead error on build completion

`--all`
Re-build all pages. The default is to only re-build pages that are new/changed since the last run.

`--builder <builder>`
Which builder to use.

Options

html | dirhtml | pdfhtml | latex | pdflatex | linkcheck | custom

`--custom-builder <custom_builder>`
Specify alternative builder name which allows jupyter-book to use a builderprovided by an external extension. This can only be used when using–builder=custom

`-v, --verbose`
increase verbosity (can be repeated)

`-q, --quiet`
-q means no sphinx status, -qq also turns off warnings

`--individualpages`
[pdflatex] Enable build of PDF files for each individual page

Arguments

`PATH_SOURCE`
Required argument

clean

Empty the _build directory except jupyter_cache. If the all option has been flagged, it will remove the entire _build. If html/latex option is flagged, it will remove the html/latex subdirectories.

```
jupyter-book clean [OPTIONS] PATH_BOOK
```

Options

`-a, --all`
Remove build directory.

`--html`
Remove html directory.

`--latex`
Remove latex directory.

Arguments

`PATH_BOOK`
Required argument

config

Inspect your _config.yml file.

```
jupyter-book config [OPTIONS] COMMAND [ARGS]...
```

sphinx

Generate a Sphinx conf.py representation of the build configuration.

```
jupyter-book config sphinx [OPTIONS] PATH_SOURCE
```

Options

`--config <config>`
Path to the YAML configuration file (default: PATH_SOURCE/_config.yml)

`--toc <toc>`
Path to the Table of Contents YAML file (default: PATH_SOURCE/_toc.yml)

Arguments

`PATH_SOURCE`
Required argument

create

Create a Jupyter Book template that you can customize.

```
jupyter-book create [OPTIONS] PATH_BOOK
```

Options

`--cookiecutter`
Use cookiecutter to interactively create a Jupyter Book template.

Arguments

`PATH_BOOK`
Required argument

myst

Manipulate MyST markdown files.

```
jupyter-book myst [OPTIONS] COMMAND [ARGS]...
```

init

Add Jupytext metadata for your markdown file(s), with optional Kernel name.

```
jupyter-book myst init [OPTIONS] [PATH]...
```

Options

`--kernel <kernel>`
The name of the Jupyter kernel to attach to this markdown file.

Arguments

`PATH`
Optional argument(s)

toc

Generate a _toc.yml file for your content folder. It also generates a _toc.yml file for sub-directories. The alpha-numeric name of valid content files will be used to choose the order of pages/sections. If any file is called "index.{extension}", it will be chosen as the first file. Note that each folder must have at least one content file in it.

```
jupyter-book toc [OPTIONS] PATH
```

Options

`--filename_split_char <filename_split_char>`
A character used to split file names for titles

`--skip_text <skip_text>`
If this text is found in any files or folders, they will be skipped.

`--output-folder <output_folder>`
A folder where the TOC will be written. Default is *path*

`--add-titles`
Whether to generate page titles from file names.

Arguments

`PATH`
Required argument

Glossary

A glossary of common terms used throughout Jupyter Book.

CommonMark
A standard syntax of Markdown that is used across many communities and projects. It is the base flavour of Markdown for Jupyter Notebook, and the base flavour for MyST Markdown and Jupyter Book.

ExecutableBookProject
The project that supports and develops many of the core tools used by Jupyter Book.

MyST Markdown
MyST
A flavour of Markdown that was designed for use with the Sphinx project. It is a combination of CommonMark Markdown and a few extra syntax pieces to support features of Sphinx, so that you can write Sphinx documentation in plain Markdown. It is one of the core technologies that Jupyter Book uses.

MyST-Parser
A parser for Sphinx that allows it to read in content that is written in MyST Markdown. It is also used by MyST-NB to parse MyST Markdown that is inside Jupyter notebooks.

MyST-NB
An extension for Sphinx that uses the MyST-Parser to parse Jupyter notebooks directly into Sphinx. This also allows users to write MyST Markdown inside of notebooks that are parsed with Sphinx. It is one of the core technologies that Jupyter Book uses.

Sphinx
A documentation engine written in Python. Sphinx supports many features that are necessary for scientific and scholarly publishing. It is one of the core technologies that Jupyter Book uses.

Binder
A free, public service for running reproducible interactive computing environments. Binder is a 100% open source infrastructure that is run by members of the Jupyter community. The underlying technology behind the Binder project is BinderHub.

BinderHub
The underlying technology of mybinder.org, BinderHub is an open source tool that runs on Kubernetes and utilizes a JupyterHub in order to provide live, reproducible interactive computing environments that users host on GitHub.

Google Colab
A Jupyter Notebook service from Google that provides access to free computing resources, including GPUs and TPUs.

JupyterHub
A core open source tool from the Jupyter community, JupyterHub allows you to deploy an application that provides remote data science environments to multiple users. It can be deployed in the cloud, or on your own hardware.

Jupyter-Cache
An open source tool for executing and caching the outputs of Jupyter Notebook content. Outputs are cached in a hidden folder so that they do not need to be included directly with your source files.

Sphinx-Book-Theme
A customized version of the PyData Sphinx Theme that defines the look and feel of Jupyter Book.

Cite Jupyter Book

Jupyter Book is developed by many people across academia and industry. Moreover, Jupyter Book is powered by an ecosystem of open source tools, most notably within the Executable Books Project.

Rather than citing each tool, we choose to use a single blanket citation for the entire project and all of its community members.

To cite Jupyter Book in publications, please use the following Zenodo reference:

DOI 10.5281/zenodo.2561065

The full author list is available by viewing the contributors graph on GitHub. To get the full picture, you should look at this for all projects within the Executable Books organization.

Change Log

v0.10.1 2021-03-02

full changelog | GitHub contributors page for this release

This is a minor update to include recent improvements made to MyST-NB

New

1. An **experimental** MyST-NB feature that enables loading of code from a file for `code-cell` directives using a `:load: <file>` option. Additional information is available in the myst-nb documentation

Upgrades

- ⬆️ UPGRADE: myst-nb v0.12.0 (#1238, @mmcky)

Maintain

- 🔧 MAINTAIN: Expand `jupytext` version pinning (#1221, @bollwyvl)

v0.10.0 2021-02-01

full changelog | GitHub contributors page for this release

This update focuses on new syntax features for MyST markdown, as well as a new configuration to enable MyST extensions. See below for more details.

New

MyST Parser version 0.13.x
The MyST-NB and MyST-parser have both been upgraded. This comes with support for new syntax and a new configuration mechanism (see below for some examples).

(see also the myst-parser changelog for more information about the syntax additions)

New `myst` extension configuration
The `myst_extended_syntax` configuration is **now deprecated**, in lieu of a more flexible extension mechanism. You may now enable individual `myst` extensions by adding them to the following section of your `_config.yml` file:

```yaml
parse:
  myst_enable_extensions:
    - <list-of-extensions>
```

See the MyST syntax extension section here for more information.

Citations and references configuration
This version comes with a version bump to `sphinxcontrib.bibtex v2.1.*`. This introduces new configuration for connecting your bibfiles (no longer using the bibliography directive), and makes the citation resolution process much more stable and dependable.
See Citations and bibliographies for more information, and the `sphinxcontrib.bibtex` documentation for more information about updates in the latest version.

TOC depth numbering
You can now set the depth of numbering (e.g., 3.2 vs. 3.2.1) via the the `numbered` flag in your Table of Contents. See the table of contents documentation for more information.

New MyST syntax

MyST Markdown substitutions**

Substitutions allow you to define **variables** in markdown, and insert them elsewhere in your document. This lets you change the variable value and have it automatically update throughout your book. This is **on by default**. See Substitutions and variables in markdown for more information.

Automatic HTML links

The `linkify` extension will automatically identify "bare" web URLs, like `www.example.com`, and add hyperlinks; www.example.com. This extension is **on by default**.

Smart Quotes

The `smartquotes` extension will automatically convert standard quotations to their opening/closing variants:

- `'single quotes'`: ‘single quotes’
- `"double quotes"`: “double quotes”

This extension is **off by default**. See this documentation for more details.

Typography replacements for common characters

The `replacements` extension will automatically convert some common typographic texts, such as `+-` -> `±`. This extension is **off by default**. See this documentation for more details.

HTML admonitions

By adding `"html_admonition"` to `myst_enable_extensions`, you can enable parsing of `<div class="admonition">` HTML blocks to sphinx admonitions. This is helpful when you care about viewing the "source" Markdown, such as in Jupyter Notebooks. For example:

```
<div class="admonition note" name="html-admonition">
<p class="title">This is the **title**</p>
This is the *content*
</div>
```

See HTML admonitions for further information. This extension is **off by default**.

Deprecations

`myst_extended_syntax` is deprecated

See above for new configuration details.

Colon fences now behave like directives

The `colon_fence` extension (replacing `admonition_enable`) now works exactly the same as normal ` ``` ` code fences, but using `:::` delimiters. This is helpful for directives that contain Markdown text, for example:

```
:::{admonition} The title
:class: note

This note contains *Markdown*
:::
```

Bibliographies no longer use a path to a bibtex file

See above for new configuration details.

v0.9.1 2020-12-22

This is a minor release to issue `v0.9` to PyPI and updates a broken link that prohibited the `v0.9.0` PyPI release action.

v0.9.0 2020-12-09

(full changelog)

This release includes a number of new features, improvements and bug fixes. There is also a new gallery of jupyter-book projects available.

New

- 👌 IMPROVE: Option to exclude every file not in the toc. (docs, #1123, @alex-treebeard)
- ✨ NEW: Enable the use of local Sphinx extension via _config.yml. (docs, #1102, @mmcky)
- ✨ NEW: Enable custom builder passthrough. This is an **advanced feature** that enables the use of additional sphinx builders via jupyter-book that may be provided by an extension. (docs, #1094, @mmcky)

HTML:

- 👌 NEW: Add `dirhtml` builder. This enables the use of the `dirhtml` sphinx builder when using jupyter book. (docs, #1092, @choldgraf)

LaTeX:

- ✨ NEW: Add `--individualpages` option for pdflatex builder. This option enables building individual (pdflatex) files for each page of the project. **Note:** Further work is ongoing to improve the styling and formatting of pdflatex output. (docs, #944, @mmcky)

Maintain

- 🔧 MAINTAIN: Pin sphinxcontrib-bibtex to ~=1.0 until compatible with recently released v2 (#1138, @choldgraf)

Upgrades

- ⬆️ UPGRADE: sphinx-book-theme v0.0.39 (#1086, @choldgraf)

Bugs fixed

- 🐛 FIX: Check for file extensions when generating toc. (#1108, @AakashGfude)
- 🐛 FIX: Export Notebook as HTML with no page-breaks. (#903, @AakashGfude)
- 🐛 FIX: Restore linkcheck to builder opts (#1051, @fmaussion)

Deprecated

- 🗑️ DEPRECATE: removing expand_sections for toc as it is deprecated in `sphinx-book-theme`. (#1073, @choldgraf)

v0.8.3 2020-10-12

This is a relatively minor release with bugfixes and under-the-hood improvements.

Bugs fixed

- 🐛 FIX: colab default is now empty #1026 (@choldgraf)

Upgrade EBT dependencies

- ⬆️ Update sphinx-book-theme v0.0.38 #1047 (@choldgraf)
- ⬆️ Update sphinx-panels pinning v0.5.2 #1044 (@chrisjsewell)

v0.8.2 2020-09-19

(full changelog)

Improved

`sphinx-panels` version bump to v0.5, which adds several new content blocks including `{tabbed}` content. #972

v0.8.1 2020-09-09

New ✨

Add `jupyter-book create --cookiecutter` (thanks to @TomasBeuzen)
This adds a `--cookiecutter` option to `jb create`, to allow users to use the Jupyter Book cookiecutter to create a book template.

The cookiecutter is suitable for more advanced users that want to create a ready-to-go repository to host their book that includes pre-populated metafiles such as README, LICENSE, CONDUCT, CONTRIBUTING, etc., as well as GitHub Actions workflow files.

Fixes 🐛

This release contains numerous improvements, to the documentation and code, to address issues noted by you guys:

- Fix issues with single document builds (e.g. pdflatex) and relative path resolutions
- Ensure `sphinx-book-theme` is loaded on PDF builds (to allow the use of the `margin` directive)
- Allow execution `timeout: -1` and `execute_notebooks: off` to be valid in the `_config.yml`

v0.8.0 2020-09-01

(full changelog)

You spoke, we listened!

Version 0.8.0 of Jupyter Book, incorporates a bottom-up refresh of the entire Executable Books Project (EBP) stack, with tonnes of bugs fixes, improvements and new features 🎉

The documentation describes all this new functionality in full detail, but below we shall try to outline the major changes and additions.

Breaking ‼️

The `jupyter-book`/`jb` executable should work almost identically to in v0.7.5, and all existing books will generally build as before (open an issue if not!).

The key change is that `jupyter-book page` is no longer available. Instead you can now pass a single file path to `jupyter-book build`, as opposed to a directory, and it will build your single page (thanks to @AakashGfude). See Build a standalone page.

Another thing to note, is that if you are using "bare" LaTeX math in your documentation, then this will only render if you activate in your `_config.yml`:

```yaml
parse:
  myst_extended_syntax: true
```

See the math documentation for details.

New and Improved ✨👌

Jupyter Book v0.8 incorporates all the great new features available by moving from:

- MyST-Parser v0.9 to v0.12 (see its CHANGELOG.md)
- MyST-NB v0.8 to v0.10 (see its CHANGELOG.md)
- This also enabled, Sphinx v2 to v3

Here's the headlines:

Windows support
  Continuous Integration (CI) testing is now run against Windows OS throughout the EBP stack. The fixes this entailed, mean that Jupyter Book can now be run on Windows with minimal issue (see Working on Windows)

Extended "Markdown friendly" syntaxes
  MyST Markdown directives offer a high degree of extensibility, to add all the features we might need to create a scientific document. However, they are not (yet) very well integrated with external editors, like the Jupyter Notebook interface. Extended syntax parsing to the rescue!

  By enabling in your `_config.yaml`:

```yaml
parse:
  myst_extended_syntax: true
```

You can access to a number of *Markdown friendly* syntaxes, which extend the CommonMark specification:

- `:::` fenced admonitions render Markdown as standard (see new style admonitions).
- HTML images are correctly handled, allowing for control of size and style attributes, and Markdown style figures extend this for captions and referencing (see images and figures).
- Definition lists are what you see here and provide a simple format for writing term/definition blocks (see definition lists).
- LaTeX math is now intrinsically supported, meaning that it will rendered correctly in both HTML and LaTeX/PDF outputs (see math and equations).

Custom Notebook Formats
Want to write your notebooks as RMarkdown, Python files, ...? Jupyter Book now supports linking any file extension to a custom conversion function, run before notebook execution and parsing. See the custom notebook formats and Jupytext documentation, which itself is written in RMarkdown!

Execution Configuration
Execution and caching of notebook outputs has been improved, to make it more consistent across `auto` and `cache` methods (`cache` execution is now also run in the notebook directory) and provide numerous configuration options, including:

- Running the execution in the local directory or in a temporary directory.
- Setting the execution timeout limit, at a global or notebook level.
- Allowing errors across all notebooks, or at a notebook or cell level.
- Removing stderr/stdout outputs or logging warnings when they are encountered
- A directive for displaying execution statistics for all notebooks in the book (status, run time, etc)

See the execution documentation for more details.

Code Output Formatting
More cell outputs are handled, including Markdown and ANSI outputs, and you can use cell metadata to set image size, style, captions and references. See formatting code outputs.

Build options and error reporting
The `jupyter-book build` command includes additional options/flags for controlling the build behaviour, such as verbose (`-v`), quiet (`-q`) and nitpick mode for checking references (`-n`). See the command-line interface documentation for more details.

sphinx-panels integration
The sphinx-panels package is now provided directly in the Jupyter Book distribution. This adds additional functionality for creating web based elements, such as gridded panels and dropdown boxes. See the Panels and Dropdowns section for details.

Fixes 🐛

Among the numerous fixes:

- Code cell syntax highlighting now works for all Jupyter kernels.
- User configuration is now recursively merged with the default configuration, and no longer overwrites an entire nested section. You can also use `jb config sphinx mybookname/` to inspect the sphinx `conf.py` which will be parsed to the builder.

More to come 👀

We have many more improvements planned, check back in this change log for future improvements.

Also please continue to provide us feedback on what you would like to see next. See our voting for new features page.

v0.7.5 2020-08-26

✨ NEW: This release introduces the new “Comments and Annotations” feature, powered by sphinx-comments. See this documentation section for further details.

Important: this version also pins the `myst-nb` dependency to v0.8. Previous versions erroneously allow for the new v0.9, which is not yet strictly compatible with jupyter-book (coming very soon!)

v0.7.0...v0.7.4

(full changelog)

Enhancements made

- ✨ NEW: Adding - chapter entries to _toc.yml #817 (@choldgraf)
- checking for toc modification time #772 (@choldgraf)
- first pass toc directive #757 (@choldgraf)

Bugs fixed 🐛

- Fix typo in content-blocks.md documentation #811 (@MaxGhenis)
- Using relative instead of absolute links #747 (@AakashGfude)
- Fixing jupytext install/UI links #737 (@chrisjsewell)

Documentation improvements 📚

- Note about licenses #806 (@choldgraf)
- Fix google analytics instructions #799 (@tobydriscoll)
- Change book_path to path_to_book #773 (@MaxGhenis)
- GitHub actions example: note about selective build #771 (@consideRatio)
- getting sphinx thebelab to work #749 (@choldgraf)
- Link documentation for adding cell tags in Jupyter from "Hide or remove content" documentation section #734 (@MaxGhenis)
- Typo fix #731 (@MaxGhenis)
- reworking interactive docs #725 (@choldgraf)
- Add documentation for Google Colab launch buttons #721 (@lewtun)
- Add note about int eq labels in math directive #720 (@najuzilu)

API Changes

- ✨ NEW: Adding - chapter entries to _toc.yml #817 (@choldgraf)
- removing config file numbered sections to use toc file instead #768 (@choldgraf)

Other merged PRs

- 📚 DOCS: Remove Issue Templates #849 (@chrisjsewell)
- 📚 DOC: document available bib styles #845 (@emdupre)
- 🐛 FIX: fixing toctree spacing bug #836 (@choldgraf)
- 👌 IMPROVE: chapters -> parts in toc #834 (@choldgraf)
- 📚 DOCS: adding information about page structure #830 (@choldgraf)
- 🐛 FIX: fixing chapters numbering #829 (@choldgraf)
- 👌 IMPROVE: improving numbered sections #826 (@choldgraf)
- 📚 DOC: update gh-pages + ghp-import docs #814 (@TomasBeuzen)
- page index #728 (@choldgraf)
- fix windows utf8 encoding 1/3 #719 (@phaustin)

Contributors to this release

(GitHub contributors page for this release)

@AakashGfude | @amueller | @bmcfee | @brian-rose | @cedeerwe | @choldgraf | @chrisjsewell | @codecov | @consideRatio | @cpjobling | @drscotthawley | @emdupre | @firasm | @jni | @jstac | @lewtun | @MaxGhenis | @mmcky | @najuzilu | @nathancarter | @phaustin | @ptcane | @samteplitzky | @tobydriscoll | @TomasBeuzen | @welcome

v0.6.4...v0.7.0

(full changelog)

Merged PRs

- Release prep #711 (@choldgraf)
- fixing topbar repo buttons #710 (@choldgraf)
- [DOC] Add info about 'remove_cell' tag #708 (@najuzilu)
- [DOC] Fix broken Jupytext link #703 (@najuzilu)
- misc doc improvements #702 (@choldgraf)
- fixing toc bug #696 (@choldgraf)
- [DOC, ENH] Shorten descriptions of clean and toc functions #693 (@pgadige)
- execution error doc #691 (@choldgraf)
- [DOC] Fix rendering of subitems in ordered list #686 (@najuzilu)
- adding configuration for source buttons #684 (@choldgraf)
- adding versions #679 (@choldgraf)
- ENH: Modify execute option for jb page command (#594 followup) #678 (@rossbar)
- fixing suffix in download files #656 (@choldgraf)
- config defaults #654 (@choldgraf)
- Additional latex newcommand example #652 (@najuzilu)
- How to reference docs and section labels #651 (@najuzilu)
- TST: minor update to reference book.pdf rather than python.pdf #650 (@mmcky)
- improving configuration under the hood #647 (@choldgraf)
- draft windows instructions #642 (@phaustin)
- [DOC] MyST cheat sheet documentation #637 (@najuzilu)
- TST: [pdflatex] Add build of quantecon-mini-example as a project style test case #636 (@mmcky)
- TST: Fix test_pdf failure. #633 (@rossbar)
- Minor update for the extra-navbar section in advanced.md #631 (@malvikasharan)
- TST: update testing requirements and adjust ignore #626 (@mmcky)
- bump peaceiris/actions-gh-pages to v3.6.1 #625 (@peaceiris)
- documenting directive keywords better #623 (@choldgraf)
- clarifying config values #622 (@choldgraf)
- add tex macro documentation #618 (@najuzilu)
- fixing docs bug in template #617 (@choldgraf)
- FIX: fix bug when latex config not specified in project for "latex_documents" #614 (@mmcky)
- adding extra footer docs #611 (@choldgraf)
- nesting notes #599 (@choldgraf)
- docs update #597 (@choldgraf)
- updating docs and fixing single page bug #592 (@choldgraf)
- add html section to tip about navbar_number_sections #590 (@amueller)
- adding custom css and js and updating docs #583 (@choldgraf)
- Devdocs #581 (@choldgraf)
- [DOC] fix contrib link in readme #577 (@amueller)
- Document use of pure html #576 (@joergbrech)
- add no_title argument to toc #571 (@amueller)
- instructions for toggle and toc docs #570 (@choldgraf)
- DOC: A few documentation fix-ups #569 (@rossbar)
- add ticks to execute_notebooks: off in docs #568 (@amueller)
- fixing flake #567 (@choldgraf)
- fix typo #565 (@amueller)
- add build to test_clean method prior to build –builder pdflatex #561 (@najuzilu)
- update vscode myst markdown extension url #559 (@najuzilu)
- [BUG] Added a returncode for make all-pdf #556 (@AakashGfude)
- [ENH] A simple {tableofcontents} directive #553 (@AakashGfude)
- clarifying qe mini example #550 (@choldgraf)
- Small edits to intro page on docs. #548 (@jstac)
- updating intro page and other docs #545 (@choldgraf)
- Update default_config.yml #544 (@Cyb3rWard0g)
- Fix broken references in docs #541 (@consideRatio)
- Corrects info on bibliography in separate file #537 (@kyleniemeyer)
- explicitly giving jupyter book install version #535 (@choldgraf)
- Add modules needed to build the example book to the instructions. #533 (@jpivarski)
- citations clean up #529 (@choldgraf)
- updating documentation #527 (@choldgraf)
- adding tests for TOC cases #525 (@choldgraf)
- adding linkcheck docs and support #524 (@choldgraf)
- update latexpdf to pdflatex in the documentation #523 (@najuzilu)

- update jb contributor link #522 (@najuzilu)
- Additional clean options #521 (@najuzilu)
- updating deploy docs #520 (@choldgraf)
- BETA: v0.7.0b1 #516 (@choldgraf)
- fixing tests and prep for new release #515 (@choldgraf)
- Rebase #496 (@choldgraf)
- migrating to new github org #493 (@choldgraf)
- adding a gallery #472 (@choldgraf)
- [ENH] Raise error if page URL does not start with os.sep #471 (@brian-rose)
- Display the Jupyter Book icon on pypi #470 (@mwouts)
- [DOC] changed circleci docs to reflect consistent job name #463 (@alexnakagawa)
- Fix links to book-html and github-pages #457 (@mwouts)
- Intronetlify #454 (@choldgraf)
- [DOC] Minor changes to documentation #452 (@rossbar)

Contributors to this release

(GitHub contributors page for this release)

@AakashGfude | @akhmerov | @alejandroschuler | @alexnakagawa | @amueller | @andrewsanchez | @asteppke | @betatim | @boazbk | @brian-rose | @cedeerwe | @choldgraf | @chrisjsewell | @consideRatio | @cpjobling | @Cyb3rWard0g | @dafriedman97 | @DavidPowell | @dhruvbalwada | @emdupre | @epacuit | @firasm | @flying-sheep | @foster999 | @gharp | @goanpeca | @grst | @hamelsmu | @jasmainak | @jgm | @jmason86 | @jni | @joergbrech | @johngage | @jpivarski | @jstac | @kyleniemeyer | @malvikasharan | @martinagvilas | @MasterScrat | @mathieuboudreau | @matteoacrossi | @mgeier | @mikdale | @mmcky | @mwcraig | @mwouts | @najuzilu | @NatalieThurlby | @ofajardo | @oscarys | @parmentelat | @peaceiris | @pgadige | @phaustin | @prabhasyadav | @psychemedia | @Racooneer | @rahuldave | @rossbar | @roualdes | @saulomaia | @TomasBeuzen | @trallard | @xldrkp | @yuvipanda

v0.6.3...v0.6.4

(full changelog)

Enhancements made

- improving the upgrade functionality #449 (@choldgraf)
- scrolling outputs #444 (@choldgraf)
- adding a page for math instructions #432 (@choldgraf)
- updating thebelab and improving code highlighting #422 (@choldgraf)
- Hide outputs and markdown cells #420 (@choldgraf)
- Adding a “test” section for pages that are hard to test w/ python #416 (@choldgraf)
- refactoring sidebar highlighting and allowing collapsed subsections #412 (@choldgraf)
- adding ability to add authors and titles #390 (@choldgraf)
- improving search functionality #374 (@choldgraf)
- Add kernel path for thebelab #189 (@joergbrech)

Bugs fixed

- Add installation module and fix circle #450 (@choldgraf)
- fixing thebelab highlighting #446 (@choldgraf)
- fixing full width content and some formatting bugs #417 (@choldgraf)
- [FIX] Fix for on-page anchor-based TOC issues #414 (@GasperPaul)
- tocfix #406 (@choldgraf)
- fixing toc #405 (@choldgraf)
- [FIX] Fixed issue with Unicode characters in TOC #400 (@GasperPaul)
- [FIX] Fix for path separator issue on Windows #398 (@GasperPaul)
- fixing issue template #393 (@choldgraf)

Other merged PRs

- moving thebelab config to within the page instead of head #448 (@choldgraf)

- only show TOC if there are headers for it #441 (@choldgraf)
- Update README.md #425 (@choldgraf)
- using thebelab latest #423 (@choldgraf)
- Update limits.md #415 (@choldgraf)
- moving some modules to pathlib #403 (@choldgraf)
- adding code structure and moving contributing guide #394 (@choldgraf)
- updating getting started guide #392 (@choldgraf)
- version bump #389 (@choldgraf)
- new release and updating instructions #387 (@choldgraf)

Contributors to this release

(GitHub contributors page for this release)

@choldgraf | @emdupre | @GasperPaul | @javag97 | @joergbrech | @melaniewalsh | @psychemedia

v0.6.0...v0.6.3

(full changelog)

Enhancements made

- adding anchors above headers #366 (@choldgraf)
- adding CSS rules for epigraphs #365 (@choldgraf)
- netlify config #359 (@choldgraf)
- Thebelab init #352 (@choldgraf)
- [WIP] Add option to clear outputs in build command #349 (@akhilputhiry)
- [ENH] Netlify Continuous Deployment #342 (@emdupre)

Bugs fixed

- css for thebelab z-order #386 (@choldgraf)
- fixing TOC auto gen bug #375 (@choldgraf)
- fixing page path link #368 (@choldgraf)
- fixing interact link bug #367 (@choldgraf)
- Update required python version #363 (@emdupre)
- fix: fuzzy matching of jupyter book versions #346 (@emdupre)
- fixing scrolling #336 (@choldgraf)

Maintenance and upkeep improvements

- fixing load ntbk function #385 (@choldgraf)
- load ntbk function #384 (@choldgraf)
- moving CSS and JS generation to their own function #381 (@choldgraf)
- making sure gemfile.lock is removed #379 (@choldgraf)
- removing unnecessary requirements #378 (@choldgraf)
- making toc gen sorted #377 (@choldgraf)
- fixing up download functionality #373 (@choldgraf)
- small refactoring of names and layout #372 (@choldgraf)
- Bump rubyzip from 1.2.4 to 2.0.0 in /jupyter_book/book_template #371 (@dependabot)
- moving to jupyterbook.org #370 (@choldgraf)
- inlining svgs and small tweaks #369 (@choldgraf)
- [fix] update docker image and documentation #364 (@emdupre)
- moving js outside of _includes if not needed #347 (@choldgraf)
- removing unnecessary clean_lines function #345 (@choldgraf)
- modularizing the bage building and beefing up single page building #344 (@choldgraf)

Documentation improvements

- DOC: intro.md: Jupyter Books -> Jupyter Book #383 (@westurner)

Other merged PRs

- Update executing.ipynb #350 (@psychemedia)
- adding better circle instructions #341 (@choldgraf)
- cleaning up circle #340 (@choldgraf)
- ghp-import in circle #339 (@choldgraf)
- removing _build artifacts #338 (@choldgraf)
- making some files optional in upgrade #337 (@choldgraf)
- adding google analytics info #335 (@choldgraf)
- updating changelog #334 (@choldgraf)
- fixing releases info #333 (@choldgraf)

v0.6.0 (2019-09-17)

Full Changelog

Implemented enhancements:

- Improve the auto-TOC function #271
- Export pages to PDF #267
- Adding popouts to the right #266
- Add option to execute notebooks when building the book #234
- Add a footer for each page #233
- adding error message context to the build CLI command #320 (choldgraf)
- Wrap `jekyll-raw` cells with {% raw %} #308 (SamLau95)
- adding popout cell #302 (choldgraf)
- adding right toc showing when there's no sidebar content #300 (choldgraf)
- adding jupytext support #280 (choldgraf)
- adding print button #279 (choldgraf)
- Updating page layout and hoverable table of contents #278 (choldgraf)
- Add a hiding topbar w/ scroll #276 (choldgraf)
- Improving TOC functionality #273 (choldgraf)
- use celltagpreprocessor to remove parts of cells and updating running code module #264 (choldgraf)
- Create footer #254 (martinagvilas)
- removing jekyll markdown templates #249 (choldgraf)
- adding simple page building #248 (choldgraf)
- HTML build step #239 (choldgraf)
- [WIP] Refactoring page layout + adding popouts and a topbar #169 (choldgraf)

Fixed bugs:

- Some small formatting issues following upgrade to master #296
- Make the PDF print work for MathJax math #285
- Code cells in plain Markdown files are rendered as raw text #283
- 'jupyter-book upgrade' deletes new references #261
- default book doesn't build properly on github - symlink error #237
- Double check installation dependencies #211
- Don't use quotes for user-entered YAML entries #305 (SamLau95)
- Set a blank excerpt for all pages #303 (SamLau95)
- fixing footer width #301 (choldgraf)
- Use CDNs for JS libraries #292 (SamLau95)
- Load thebelab asynchronously #291 (SamLau95)
- Fix missing </div> if page.interact_link is false #290 (SamLau95)
- fixing jupytext markdown inconsistencies #288 (choldgraf)
- fixing double math printing #286 (choldgraf)
- fixing up print functionality #284 (choldgraf)
- making a download PDF button appear on all pages #282 (choldgraf)
- fixing the TOC function #270 (choldgraf)
- [FIX] Fix references being deleted with jupyter-upgrade #263 (martinagvilas)
- fixing links #260 (choldgraf)
- adding instructions for build #257 (choldgraf)
- fixing pypi description #256 (choldgraf)
- version fix #250 (choldgraf)

Closed issues:

- Jupyter Notebook can't close a running notebook #317
- Building book fails with a jinja2.exceptions.TemplateNotFound error #310
- serving non- md/ipynb content #295
- Broken links #259
- Missing file error #253
- Multicursor sometimes only deletes one line when there's spaces #251
- Release summary for v0.6 #331

Merged pull requests:

- fixing releases info #333 (choldgraf)
- dev0 bump #332 (choldgraf)
- bumping version for release #330 (choldgraf)
- fixing up execute docs #328 (choldgraf)
- updating documentation for build #326 (choldgraf)
- adding miniconda-based test #324 (choldgraf)
- removing outdated FAQ entry #319 (choldgraf)
- Changes to make work with `conda skeleton pypi jupyter-book` #315 (krinsman)
- Update Docs to include alternative method for building Jekyll locally. #313 (krinsman)
- Omit entries w/o URLs and external links from TOC #309 (SamLau95)
- documenting execution functionality #299 (choldgraf)
- adding beta label to features #294 (choldgraf)
- adding a sample CircleCI build config #293 (choldgraf)
- footer width #287 (choldgraf)
- adding help entries #277 (choldgraf)
- Circlecibot #269 (choldgraf)
- factoring out page module #265 (choldgraf)
- summary not large image for twitter #247 (choldgraf)
- twitter share #246 (choldgraf)
- updating changelog #245 (choldgraf)
- version bump to dev #243 (choldgraf)

v0.5.2 (2019-07-26)

Full Changelog

Implemented enhancements:

- implement removecell for markdown cells #192

Fixed bugs:

- yaml.load() is unsafe #230
- Update documentation links that are broken #224
- pip installation: jupyter-book or jupyter_book? #184
- Figure out why Gemfile and Gemfile.lock are causing issues #154
- Page turn link URLs missing .html #140
- Errors when running on Windows #137
- Docs aren't being updated from master #136
- toc help is incorrect #132

Closed issues:

- Update docs about how to hide code cells #240
- Is it possible to export Word, PDF and other formats? #228
- Make it clear in the documentation that the full docker path needs to be specified, not the relative path #220
- Why do we have "content" directory inside the "_site"? #219
- mismatch in docs and functionality #214
- Double check that `generate\_toc.py` is getting copied properly #210
- jupyter-book cli does not work as stated in the getting started guide #208
- Unable to install jupyter-book in conda environment #206
- Error if kernelspec missing #195

- Clarify the CLI help statements #146
- jupyter-book upgrade also modifies requirements.txt #130

Merged pull requests:

- bumping version and adding CLI for version #242 (choldgraf)
- Release fixes #241 (choldgraf)
- Markdown exporter in Python instead of the CLI #235 (choldgraf)
- Change text in Markdown cell to correct URL. #227 (habi)
- [DOC] Clarify full vs relative path in container build instructions #226 (emdupre)
- Ensure UTF-8 Encoding When Building Book #225 (cczhu)
- version bump to dev #218 (choldgraf)
- version bump for bugfix #217 (choldgraf)
- fix doc mismatch for "make build" #216 (thammegowda)
- make scripts dir as a module, to be included by `find\_packages\(\)` of setuptools #215 (thammegowda)
- add jupyter book to template requirements #209 (choldgraf)
- updating hidecode tag word and allowing total HTML removal #207 (choldgraf)
- Ask if kernelspec exists in metadata #197 (joergbrech)
- changelog and version bump #194 (choldgraf)

v0.5 (2019-05-13)

Full Changelog

Implemented enhancements:

- Markdown guide refers to Highlightjs but Rouge is used #183
- Get codecov working #153
- Add thebelab button to every code cell #117
- Add an option / config for analytics tracking? #115
- Add support for nbinteract #82
- Scrolling in subtitle column #173

Fixed bugs:

- Page turn links to external sites are broken #186
- Code cell pre-wrap causes split lines #182
- problems building with images #124
- Problems with local build instructions on Mac (and perhaps other 'nix platforms) #123

Closed issues:

- Plots not showing in ipynb files #179
- jupyter-book is missing from the binder requirements #166
- Binder and Thebelab not working for demo book #155
- Errors building HTML #152
- Non `--demo` option seems to fail #120
- Error on notebook.py with inconsistent and DEFAULT_STATIC_FILES_PATH. #108
- add requirements to setup.py #105
- Add a CONTRIBUTING.md #99
- Explore using a cookiecutter for the site #87
- Use a Python CLI instead of Make #42
- Autogenerate toc #40
- Force permalinks to be lowercase and replace space and '_' with '-' #35
- Extra buttons to support #32
- Use a submodule for notebooks folder #14
- Jupyter Book version v0.5 #175

Merged pull requests:

- Release #193 (choldgraf)
- Update local install instructions #190 (mwcraig)
- fixing some css bugs #188 (choldgraf)
- [FIX] Issue #137 Errors when running on Windows #187 (stafforddavidj)

- fixing content root notebook problem #181 (choldgraf)
- wrap code even if it has no language #180 (alexmorley)
- removing custom TOC code #178 (choldgraf)
- Make right hand toc scrollable. #176 (alexmorley)
- adding cell tags metadata #171 (choldgraf)
- make sure jupyter-book is included in the binder build #167 (joergbrech)
- fixing thebelab css and splitting off the interactive notebooks #165 (choldgraf)
- [ENH] Add thebelab button to every code cell #163 (joergbrech)
- making tags for removing cells not text #162 (choldgraf)
- hide cells updates #161 (choldgraf)
- updating css to match input and output #160 (choldgraf)
- Update config.yml for code coverage #159 (choldgraf)
- codecov activation #158 (choldgraf)
- Update requirements.txt #157 (choldgraf)
- binder links to gh-pages now #156 (choldgraf)
- Fix broken notebook links #150 (mwcraig)
- fixing thebelab and circle build #149 (choldgraf)
- removing build folder #144 (choldgraf)
- maintaining docs for site structure #142 (choldgraf)
- Build update #135 (choldgraf)
- Release guide #131 (choldgraf)
- [MRG] Refactor argparse #129 (jasmainak)
- Add badge for coverage #128 (jasmainak)
- [ENH] Initial commit of dockerfile, updated docs #127 (emdupre)
- MAINT: make jupyter-book conform to pep8 #126 (jasmainak)
- linking minimal folder #122 (choldgraf)
- adding nbinteract support #119 (choldgraf)
- update link of the markdown version of guide #118 (cnydw)
- [ENH] add google analytics option #116 (joergbrech)
- improving the non-sidebar layout and toc script #112 (choldgraf)
- [doc] contributing guidelines #111 (emdupre)
- Cssfix #109 (choldgraf)
- source dependencies from requirements.txt #106 (Zsailer)
- Adding download links for the notebook files #104 (choldgraf)
- fixing thebelab keyboard shortcuts behavior #103 (choldgraf)

v0.4.1 (2019-02-09)

Full Changelog

Closed issues:

- Disappearing None #98
- Inquiry: Plotly interactive plots in a Jupyter Books? #93
- iframe not rendering #91
- thebelab uses the wrong kernel #90
- An option to embed a link on sidebar logo #77
- Scrollbar overlaps with TOC table (on Linux) #75
- Unwanted leading white space at the beginning of code block #73
- Standard badges rendering too big #65
- Give a shout-out to bookdown #63
- Make it clearer how to customize the look and feel of the site #61
- Recommend a way to make citations #60
- Highlight active section in right sidebar #55
- Allow people to put YAML in their content files #51
- Site Search #39
- Conversion from old system to new #37
- Support “versions” of a book #31
- Book is not rebuilding #29
- Feature request: right-side navbar auto-scroll #24
- generate_summary_from_folders doesn't output prefixed numerals #23

Merged pull requests:

- fixing thebelab keyboard movement #102 (choldgraf)
- Styles #101 (choldgraf)
- Update FAQ with guidelines for Plotly #97 (mathieuboudreau)
- set kernelOptions for thebelab based on notebook's metadata #92 (joergbrech)
- [WIP] adding a CLI to generate books #89 (choldgraf)
- Fix very small typo #88 (mwcraig)
- adding update instructions #86 (choldgraf)
- adding search functionality and external links in sidebar #85 (choldgraf)
- adding codemirror theme config #84 (choldgraf)
- ignoring gh-pages for circle #81 (choldgraf)
- transferring over book to new owner #80 (choldgraf)
- sidebar logo link #79 (choldgraf)
- improve codemirror syntax highlighting #78 (choldgraf)
- updating sidebar css to be more minimal #76 (choldgraf)
- Rebuild file if source file has a newer time stamp #74 (gaow)
- Deploy to github.io using circle and update docs to reflect this #69 (choldgraf)
- updating requirements for binder #67 (choldgraf)
- adding thebelab buttons and some other updates #66 (choldgraf)
- adding citations support #64 (choldgraf)
- adding collapsible code blocks #59 (choldgraf)
- highlighting to right menu bar #58 (choldgraf)
- updating install instructions to use conda #57 (choldgraf)
- adding ruby to circle #56 (choldgraf)
- Make the sidebar stay on the page during scroll #54 (ReventonC)
- adding mini module and yaml splitter #53 (choldgraf)
- adding a default license to the book #48 (choldgraf)
- fixing the internet js to not use jquery #47 (choldgraf)
- Delete duplicated link in readme #46 (consideRatio)
- Typo - does it matter? #45 (consideRatio)
- Fix broken links #44 (consideRatio)
- updating guide to latest version #43 (choldgraf)
- adding tests and some more command-line options #41 (choldgraf)
- Split requirements into build and run #36 (matthew-brett)
- moving notebook images folder to build #34 (choldgraf)
- fixing interact link paths #33 (choldgraf)
- Refactor textbook generator to check redirects #27 (matthew-brett)
- A blank target URL here would be nice. #15 (arokem)

v0.2 (2018-10-23)

Full Changelog

Closed issues:

- Feature request: Search Bar #25

Merged pull requests:

- Update to new build system #30 (choldgraf)

v0.1 (2018-10-20)

Closed issues:

- Calling newer version of nbclean than is available on pypi #21
- Changing MathJax Size of only Blocks #19
- Change Color of Links #18
- Enhancements to pull from the DS100 textbook #17
- Hidden Code Blocks #13
- MathJax Rendering Issues #12
- Generate textbook not finding README.md #11

- Fork and clone may not be the best workflow #10

Merged pull requests:

- Fix dollar escapes at beginning of line #28 (matthew-brett)
- Add pip requirements file #26 (matthew-brett)
- Made textbook_folder an optional input #22 (jmason86)
- adding advanced section #16 (choldgraf)
- adding copy buttons #9 (choldgraf)
- fixing c3po #8 (choldgraf)
- adding intro material #7 (choldgraf)
- adding sidebar and header inferring #6 (choldgraf)
- image center and max width #5 (choldgraf)
- updating content width #4 (choldgraf)
- stylistic improvements to textbook setting #3 (choldgraf)
- updating chapter links #2 (choldgraf)
- Build missing files #1 (choldgraf)

** This Change Log was automatically generated by github_changelog_generator*

By The Jupyter Book Community
© Copyright 2020.