TeachBooks Manual

van Woudenberg, T.R.; Lanzafame, R.C.; Kirsch, J.A.A.; Jungbacker, C.A.A.; Pols, C.F.J.; den Ouden-van der Horst, D.; Slingerland, I.C.

DOI
10.5281/zenodo.15100848

Publication date
2024

Document Version
Final published version

Citation (APA)
van Woudenberg, T. R., Lanzafame, R. C., Kirsch, J. A. A., Jungbacker, C. A. A., Pols, C. F. J., den Ouden-van der Horst, D., & Slingerland, I. C. (2024). *TeachBooks Manual*. GitHub. https://doi.org/10.5281/zenodo.15100848

TeachBooks Manual

Contents

This manual is primarily designed for and by teachers for use in education, but should be a useful resource for anyone interested in creating and collaborating on jupyter{book}. Our aim is to provide a simple way to start book-making for new users (it only takes 10 clicks!) through advanced usage for experienced users. We hope you find this resource useful and refer back to it often.

TeachBooks originated at Delft University of Technology in the Netherlands, and some of the material in this Manual reflects tools and resources specific to TU Delft. With the exception of a few tools for which we have educational licenses at Delft, everything in this Manual is open source. Despite this, we are happy to grow our community beyond Delft and welcome contributions from all users.

For more information about TeachBooks, visit https://teachbooks.io/. Do not hesitate to reach out via email at info@teachbooks.io or contribute on GitHub via discussions, issues or pull requests.

Happy book building!

How to Use this Manual

There are several "parts" to the manual:

- **Your First TeachBook!** introduces the essential platforms and workflows, and contains a *workshop* that can be completed independently or as part of a TeachBooks training. We recommend even experienced Jupyter Book users go through this material to better understand the TeachBooks vision on how to make books collaboratively, as this drives much of our tool development. The workshop is designed to be completed in 1-2 hours and does not require any prior knowledge or installation (just a GitHub account).
- **Getting Going!** begins with an overview of *User Types*, which are used to help understand which parts of the Manual are most relevant to you. Detailed software

installation instructions, book and team setup and workflows are also provided in this part.

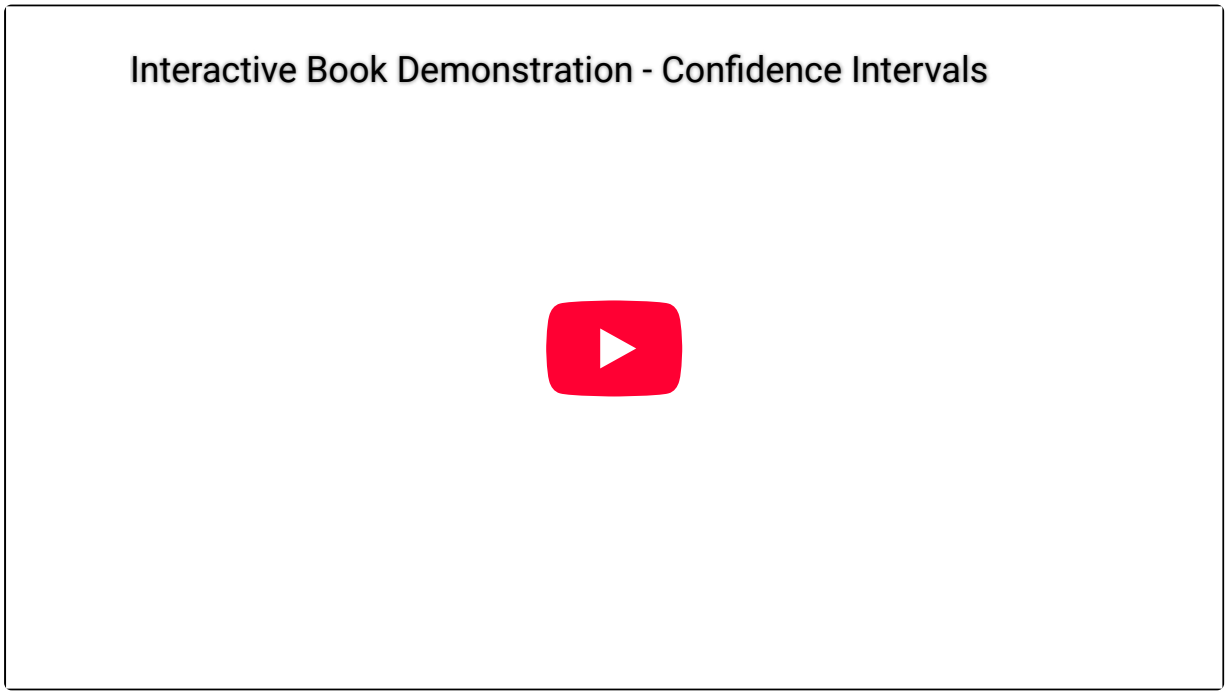
- The **Features** part describes a suite of tools that are useful for teachers, many of which are developed by TeachBooks contributors specifically for use in education. Many of these tools are illustrated in the **Examples**.
- A few special tools are included in the **Editing Tools** that are useful when writing content.

See the final chapters of this book (under **Miscellaneous**) for additional information about References, Credits, etc.

What is a TeachBook?

Contents

Never seen a TeachBook before? Here's a small demo of our MUDE-book!



So what are you looking at? It's a website generated with the Jupyter Book package. Various extensions have been added to improve the student-experience, especially for technical topics! The actual website you're looking at now is a Jupyter Book. With TeachBooks we aim to make it easy to collaboratively use it, even if you've little experience with the software package which is part of it.

Three Key Ingredients

To create a Jupyter Book, three key ingredients are needed:

1. A **configuration file** to define functionalities, `_config.yml`

2. A **table of contents** file to list which files should end up in the book, `_toc.yml`, as well as define its structure
3. **Content** for your book! Text-based files which allows you to embed figures, videos, math and interactive elements.

The files are typically organized together in a files structure like this:

```
└── my_book_directory/
    ├── README.md
    ├── requirements.txt
    └── book/
        └── _config.yml
        └── _toc.yml
        └── contents/
            └── intro.md
            └── chapter1.md
            └── chapter2.ipynb
            └── images/
                └── figure1.png
```

Note the arrangement of the 3 key book ingredients in a subdirectory called `book/`; this is where you should focus when you are just getting started with TeachBooks and are making your first book. Additional files can be used to control the book-building process, and these are typically stored *outside* the `book/` directory to keep the book contents separate. For now, all you need to understand that everything needed to create a book can be collected in a single top-level directory (`my_book_directory/`, in this case), and this directory is what becomes the collection of files in our Git **repository** (described elsewhere).

Text-based files? yml? What is that?!

Text-based files are digital files on your computer that you can open in a text editor and read directly. There are many file extensions used for text-based files, a few common ones related to Jupyter Book are: Markdown (`*.md`), Jupyter Notebooks (`*.ipynb`), and YAML (`*.yml`).

YAML (or YML) is a text-based file format that is primarily used to store data. It is useful because it is easy for humans to read and write, and easy for machines to *parse* and generate., which is why YAML is often used for configuration files. Here is an example of a YAML file...can you tell what information is being stored?

```yaml
parts:
  - caption: Your First TeachBook!
    chapters:
    - file: intro/book.md
    - file: intro/workflow.md
    - file: intro/template.md
    - file: external/next.md
  - caption: Getting Going!
    chapters:
    - file: ...
```

Description of the YAML file

You can see quite clearly that a number of files are listed, and that they are organized into chapters and sections. This defines the structure of the book—you can confirm this by comparing to the left sidebar, as this book is where this example YAML snippet came from!

How is a book made?

A book is made by by writing content in text-based files such as Markdown (`*.md`), Jupyter Notebooks (`*.ipynb`), etc. After that, software is used to "parse" these files and create the final book (e.g., Jupyter Book). Fortunately when getting started you don't need to worry about the software because TeachBooks has set up GitHub tools for you that take care of this process automatically! The process is illustrated in the diagram below.

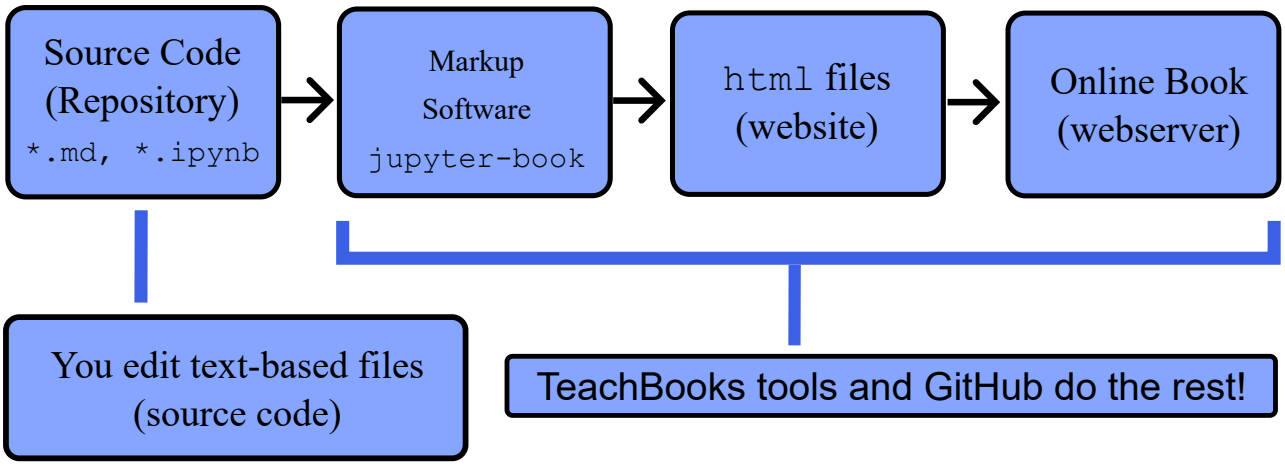


Fig. 1 The process of building a book with TeachBooks. When just getting started, TeachBooks tools enables you to focus on what matters most: learning to write content for your book!

Familiar with LaTeX? There are similarities!

Have you ever used LaTeX? It turns out there are a lot of similarities with Jupyter Book. Here is a quick list:

1. Writing "source code" in text-based files (i.e., `*.tex` versus `*.md`, `*.ipynb`, etc) that is generally easy to read without the markup and uses special functions to format rich document objects (e.g., tables, figures, etc).
2. The "book" is created by using a piece of software that parses the source code and creates the final document (a `*.pdf` versus a `*.html` for Jupyter Book).
3. Creating the document structure using a list of files that contain source code that automatically generates the Table of Contents and Index..
4. Sometimes you can spend more time than you like troubleshooting "bugs" that turn out to be simple syntax errors.

If you've used LaTeX before, it will be relatively easy to learn to use a Jupyter Book.

Why are Git and GitHub Important?

The fun part is the collaborative aspect of TeachBooks! However, the amount of software required *when doing this by yourself* can be intimidating. Git and GitHub are tools used by TeachBooks to make this process as smooth as possible for newcomers.

There are three reasons why Git and GitHub are important for TeachBooks:

1. **Version control**: Git is a version control system that allows you to track changes to your files over time. This is especially useful when you're working with others on a book, as it

allows you to see who made what changes and when.

2. **Collaboration**: GitHub is a company that provides a wide array of online tools. These *augment* the version control features incorporated directly in Git. In short, tools at `github.com` allow you to share your files with others and work on them together. This is especially useful when you're working on a book with multiple authors, as it allows you to easily share your work and collaborate.
3. **Automation**: GitHub provides a number of tools that can help automate the process of building your book. TeachBooks has carefully developed a number of these tools specifically to support teachers and students in the process of creating and sharing educational content.

What's next?

The following pages will introduce you to a standard "workflow" for editing a book, which is focused entirely in the "source code" of a book (the first box in the diagram above). After we cover this, you'll be ready to make your first book!

How can you do it?

Contents

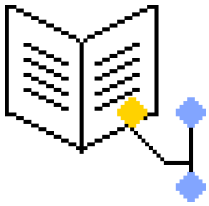
The fun part is the collaborative aspect of TeachBooks! Because all the files are text-based, they're well suited for using git. That allows us to use the version-control principles of git to its best!

Assuming you have a book (well make that happen soon!) and you're ready to collaborate, the process is as follows:

1 get an idea

You have an idea to improve a book or add some content! Let's make that happen! Share you idea with the book-authors to let them know you're starting something! Maybe someone else has some ideas on it as well. Later on, you can use GitHub `Issues`, `Projects` and `Milestones` to keep track of this.

2 create your version of the book



Let's create a space for you idea to form. You can create your own version of the book, visible for everyone to see. On GitHub, you'll use `branches` or `forks`.

3 edit the book

Creativity activated, let's start adding content to your book! You'll add text, images, videos, math, interactive quizes, coding, widgets, etc. As editing the book is not directly done in a graphical interface with buttons like Microsoft Word, you'll need to learn a bit of syntax. The MyST syntax cheat sheet will be added to your bookmarks. The changes you'll make have to be added to the GitHub version control system, creating `commits` on the your git `branch`.

4 check changes

You've made a change, how does is look like? Your raw file have to be parsed. You can do that yourself or use the tooling provided by use in a GitHub `Action`. As soon as you uploaded (`pushed` in Git language) your changes to GitHub, it'll create a nice looking website for you!

5 repeat, edit and check

You'll make mistakes and you'll want to alter content... many, many times... As long as your idea is not ready to be shared with other, keep iterating step 3 and 4 until it's perfect!

6 submit for review

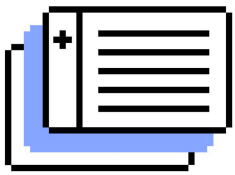
You're done! Great! Now let's see what others think of your contribution. You'll submit your changes to the original book, in GitHub language this is called a `Pull request`. Other will be able to review, adapt and eventually `merge` your version into the book!

I hope that all of this seems fun! Let's continue on making this happen!

It only takes 10 clicks!

Contents

Now that you've been introduced to the basics of a Jupyter Book, let's make one! The TeachBooks Team has put together a set of tools to make the book-creation process as smooth and automatic as possible. This page guides you through the process of creating a book, followed by a series of exercises that are designed to help you get used to the syntax and structure of a Jupyter Book, as well as the tools and essential workflows associated with the editing, checking and review processes.



1. Create a GitHub account

TeachBooks relies of freely available services from GitHub to store files online, carry out the book-building process and host your book as a website. All you need to get started is a GitHub account (not counted in the 10 clicks 😁).

> **Exercise 1**
>
> If you don't already have one, create a GitHub account here by visiting github.com/signup.

2. Use the template—it's really just 10 clicks!

Now, let's make a book: we've made a template on GitHub which carries out *most* of the technical steps required to create a book and put it online. Instructions for using the template are in the `README` of the TeachBooks Template repository on GitHub, which is by default visible on the repository homepage.

Exercise 2

Go to the TeachBooks Template repository on GitHub and follow the instructions to create your book. At the end, it should be visible at `<your_username>.github.io/<book>`. When finished, come back to this page.

3. Let's practice!

Getting used to the syntax of writing in a Jupyter Book can be a bit daunting at the start, not to mention becoming familiar with the various tools and workflows required. To help make the learning curve somewhat easier to travel, a number of exercises covering the TeachBooks workflow have been prepared for you. The exercises are designed such that no prior experience with any tools or programming is required, as long as you have read the preceding introductory pages.

The exercises on the TeachBooks Workflow can be found in your book at `https://<username or organiszation_name>.github.io/<repository_name>/exercises/exercises` (case sensitive) or at https://teachbooks.io/template/exercises.html.

Template

Search Ctrl + K

Contents

Tip

Maybe you're already comfortable with Git, GitHub and the concept of using software to parse text-based files and create marked-up documents. In that case, can you answer the following questions (refering to the exercises):

1. Can you add some content to the intro page?
2. Can you add the file named `file_to_be_added_to_toc.md` to the book website?
3. Can you edit the repository url defined in the `_config.yml`, change the title shown below the logo and change the author as shown in the footer?
4. Can you make a new branch of your book with an additional file `file_on_new_version.md` and view it online?
5. Can you merge your branch into `main` with a pull request?
6. Can you fork someone else's repo, and suggest a change in a pull request?

If something is not clear, dive into to the relevant exercise page.

You Know the Basics: What Next?

You've seen what a JupyterBook is, how TeachBooks tries to help you with making those, and what the potential is in teaching. So what's next?

In this manual we explain how to get going depending on the way you want to use it. If you successfully followed the steps until here, you'll be able to make book as a 'user type 3'!

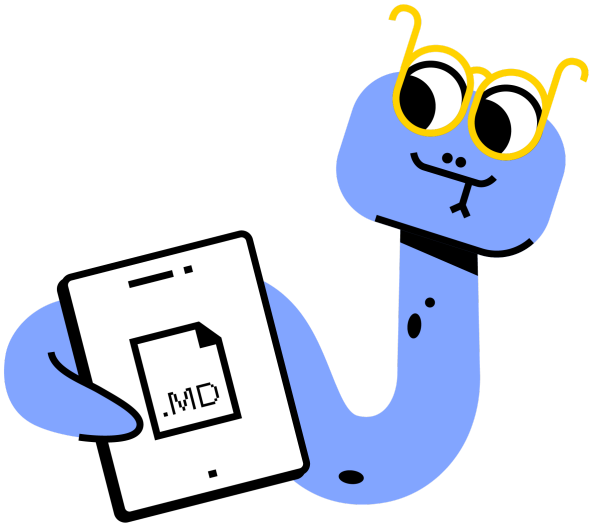


Fig. 2 User type 3

In the next part of this manual 'Getting going!', we'll explain which options you have as more advanced user, how to collaboratively edit books en how to organize you team, tools, websites and copyright-considerations.

Furthermore, in the 'Features' part we'll collect a suite of existing open-source software to ease the editing process, add functionalities, improve styling and improve the student-experience. Finally, the 'Examples' page contain some example pages of some book.

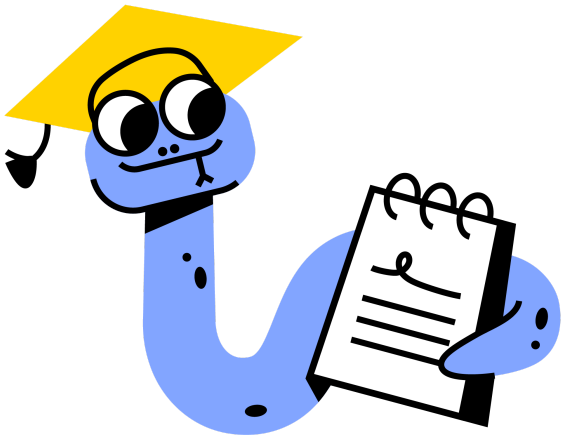
We hope you enjoin making books with TeachBooks. If you encounter any problem, please reach out to us on email or our GitHub Discussions. We'd be happy to collaborate with you or simple receive some feedback, examples, cool ideas. Happy book building!

User types

Contents

Different types of users will interact with the website or the Jupyter Books differently and will therefore need to know more or less about the composition of the book. For now we will differentiate 5 types of users.

User type 1: Students providing feedback via the book website



There are multiple tools of providing feedback in the built book:

1. using the issue button in the top-right corner to directly create issues visible to the editor team.
2. Using the hypothesis-extension which allows public/private highlighting and annotating parts of the book.

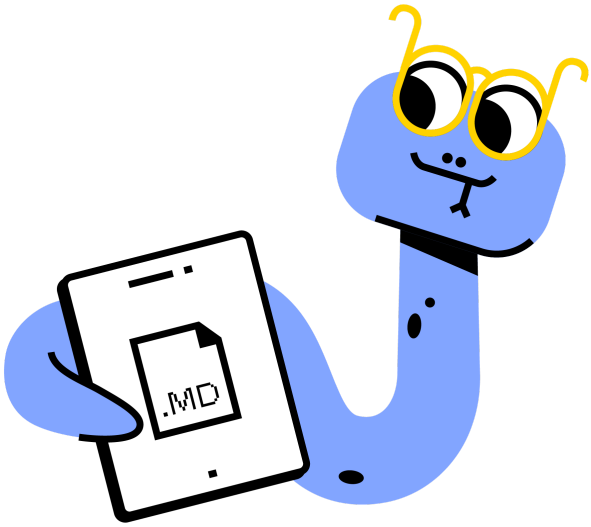
3. Using the Utterances-extension which allows commenting on pages for books which are hosted on GitHub visible for everyone. Those comments are converted to issues on GitHub for the editors to handle.

User type 2: Colleague providing feedback on a draft website

In a similar manner as students, colleagues and team members can leave content-related comments on the website through extensions or GitHub/GitLab features (e.g., Issues). In this way, multiple teachers can give input in the draft of the textbook and course material without requiring a complete understanding of Jupyter Books or GitLab/GitHub. The responsible teacher (or student assistants) can then make adaptations to the book as recommended by their colleagues.

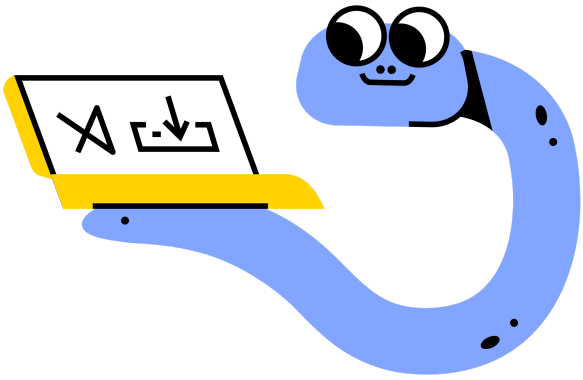
User type 3: Colleague making adjustments

via the browser



This type of user requires only very basic knowledge in GitLab/GitHub in order to make changes to the website themselves. Most importantly, there is no need to install any software! Making adjustments to a file only requires the user to be able to login into GitLab/GitHub and edit .md or .ipynb files. The changes can be done directly in an existing branch, or, if unsure, a new branch can be used for the changes then the set-up and merging can be completed by someone more familiar with GitLab/GitHub. Files can be edited individually or many-at-a-time using the VS Code-style browser application.

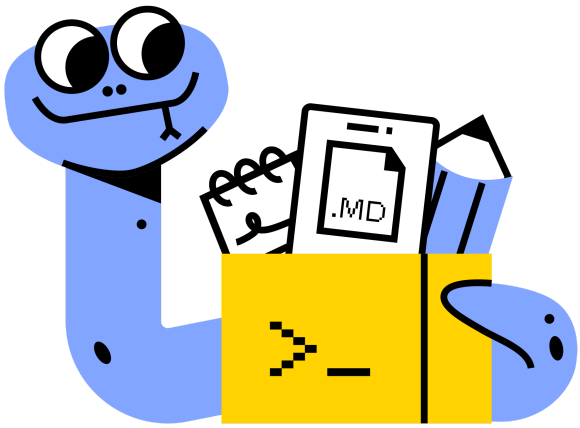
User type 4: Colleague making adjustments locally



User 4 is a hybrid between type 3 and 5. Basically this user is making changes by editing the .md files locally using, for example, VS Code. Therefore, this user requires a little bit more knowledge on git (think of pulling and pushing changes) but their main focus remains to make adjustments to the content of the book by editing .md files whereas user type 5 has more knowledge to make changes to python files the book as well. This user can build the

book online and therefore does not need to have special software installed besides a text editor (e.g., Python is not required). Using this skills, this user type can take the role as editor, managing more than just book content.

User type 5: Colleague making adjustments and incorporating them in the book locally



This user has more knowledge in GitLab/GitHub than the previous user and can take care of branching and merging directly. They know how to work with software for editing .md and .ipynb files. They can use the Deploy Book Workflow online to view their changes in the book and coordinate peer review, and/or build the book locally. This user can easily take the role as editor, and might even add additional features to the book. If you do so, please share those on the TeachBooks-community so that other people can benefit and contribute to your tooling.

Note

At this point, you might already be trying to see which user type fits you the best. We are aware that the lines between user types can be a bit blurry, especially between type 3, 4 and 5. If you've no experience in git and programming, we advise you to start as user type 3, if you're a bit more experienced you can consider skipping that step. In case you're (1) a user type 3 and want to dabble in git or (2) a user type 4/5 and want to make some quick edits with only a few clicks, then you might consider using a more advanced online editor in GitHub (GitHub Dev) as explained further on in Collaborative book-editing > Edit > User type 3 > GitHub > Using VS Code in your browser.

Collaboration tool: Git

Contents

You will need to interact with Git to collaborate on a TeachBook, even if you are working by yourself! You've been introduced to it in the exercises in the template and you'll be guided through all of the steps in detail in Collaborative book-editing, but what is this "thing" that sounds suspiciously like something only a computer scientist would want to use?

> **Tip**
>
> Perhaps you are wondering: *Why a new tool? Why not something I already know like Microsoft Word and Track Changes? Or something like OneDrive or Dropbox?*
>
> - Git is 100% free, and online tools like GitHub extend this functionality for free.
> - Git is explicitly designed for text-based files, which is how the content of a TeachBook is written.
> - You will never have to work with many versions of the same file again (e.g., no more "My_Doc_FINAL_v3_RL_again_final.docx"!).
> - Online tools like GitHub provide an easy way to backup your work *and* to allow others to see it.
> - Online tools like GitHub also provide a great way to collaborate, for example: discuss changes to your content, allow others to make suggestions for improvement and fix software issues.
> - Online tools like Zenodo allow for long term (decades!) preservation of your work and the ability to update your work with version numbers, dates and DOI's.
>
> We don't deny that it can take some time to get familiar to the new terminology and workflow associated with Git, and even longer to feel comfortable with it. However, we *promise* that if you dedicate a little bit of time to learning about this tool it will make your new life creating online interactive textbooks much easier and more enjoyable.

What is Git?

Git is a version control system (VCS), used by a wide variety of engineers and software developers to work on projects in parallel together. It provides multiple benefits such as tracking changes to files, working side by side with other people, and the ability to rollback to previous versions of files without losing track of newer changes. It is a free and open sources software. It is especially useful for TeachBooks because it is explicitly designed to handle text-based files, which are essential for writing the contents of a book.

What is GitHub?

GitHub is a software service that is accessed primarily via a web browser at `github.com`; since 2018 it is owned by Microsoft. It is the most well-known cloud-based Git provider and provides a lot of functionalities for free for open source projects, and even more free services for education. Via a service called GitHub Pages, it also allows you to host websites on `github.io` for free eliminating the need for your own web server (however, you don't have full control over the server setup and operation).

For new TeachBooks users, we advise you to use GitHub. In addition, we recommend you start by making all of your respositories public (unless they contain sensitive information); this will make it easier for others to collaborate with and

help you if you get stuck!

What is GitLab?

GitLab is a cloud-based version control system built around Git; it is a competitor of GitHub, as the two provide many of the same services (but note the names of things can differ, for example, a *Pull Request* on GitHub is a *Merge Request* on GitLab). GitLab provides a lot more features to extent Git such as Issues, Merge Requests, CI/CD pipelines, etc.

For TU Delft employees: TU Delft has a license to use GitLab on our own local webservers—this means that all of the files are stored digitally on the TU Delft campus, rather than some unkown webserver that could be physically located in an undesirable (physical) location. This is also why we have our "own" GitLab located at `gitlab.tudelft.nl`, rather than the "normal" GitLab at `gitlab.com`. One reason we don't recommend beginning TeachBooks users at TU Delft start with the TU Delft GitLab is that it does not have an automatic website hosting service set up; instead, it must be manually set up on a TU Delft web server (this is straightforward to do, for example, by BSc students from the Computer Science faculty). GitHub Pages provides this service for free and has proven to be very reliable, so start there!

Additional lessons

Software carpentry offers git-lessons if you'd like to learn more than just it's specific use for TeachBooks. The lessons are available online and TU Delft offers these as part of a Software Carpentry workshop

Which git provider to choose

Contents

The setup of tools and URLs is dependent on agreements you've made on collaboration.

Choosing between GitHub and GitLab depends on multiple criteria. GitHub provides more functionalities, but you might prefer the TU Delft-closed GitLab system if you're a TU Delft employee.

Book URL

Github allows you to host your book on the GitHub server using GitHub pages (to be recognized by the `<organization/username>.github.io/<book>` url, for example the template book), which takes all the steps of hosting out of your hands. Next to GitHub-provided URLs, you can set up a custom owned URL, although this requires some additional skills on a domain which you should own.

If you're part of TU Delft, you can put your book in the TUDelft-Book organisation, which allows you to host your book at `oit.tudelft.nl/<book>`. If you've a private book which is part of the GitHub Enterprise TU Delft and is using SSO, a random url is generated `<random>.github.io/<book>`. For courses we advise you to create a organization with the course name so that the URL represent that course.

On TU Delft's GitLab you need a webserver, which was offered by TeachBooks in the past (to be recognized by the url `teachbooks.tudelft.nl/<book>`), and is still offered by the TU Delft OIT team (to be recognized by the url `interactivetextbooks.tudelft.nl/<book>`).

Real-time editing book

Editing the book can happen both locally as online on GitHub/GitLab. If done locally, the book can be generated and viewed locally too (typically takes around 1 minute) but this requires you to be at least user type 4 or 5 depending on the book content. The built book can also be viewed online separate from the released version.

On GitHub we developed a automatic process which builds the book and publishes it online in a very flexible way (publication of multiple version of the book, insights in book building errors, parallel so fast build, custom urls in subdirectories). For a private repository, there is an extensive but limited amount of actions-minutes required for published your book. If this proves to be a limit, consider applying for Teacher benefits or joining the GitHub Enterprise of TU Delft.

On GitLab a webserver is required to process book-changes online. Both we and other people at TU Delft have a simplistic workflow which can be used when you're set up on those Git environments and webservers. This simplistic workflow doesn't build all branches, is not easily adaptable, doesn't cache environments, doesn't give a visual summary, and doesn't allow for parallel processes (for every build a runner needs to be assigned for which there's typically only one available). If you want to use GitLab but still want to make use of the GitHub workflow, you can mirror your repository to GitHub. For the TU Delft OIT webserver you're required to publish your book more officially, it cannot be used for viewing your book online in an active editing-phase since constant copyright checks have to be performed. The amount of computing power available is dependent on how you set the server yourself.

Setting up book repository and website

On GitHub you can start right away with a git environment and online book using our template without the need for any webserver setup!

On GitLab you can set up your own git environment, but you need to be given access by TU Delft OIT to view your build book online if you don't have your own webserver-setup.

Book access with SSO

When you're releasing your book on a server on which you're in control (connected to GitLab or GitHub), you can set up SSO login for visitors of the website. In the past, this was arranged for for TU Delft employees at teachbooks.tudelft.nl. However, this is depreciated.

If you want the same functionality with GitHub pages, your book should be part of the GitHub Enterprise of TU Delft. This SSO login is a bit different as you're required to give access to specific accounts. Furthermore, the url of you're book on GitHub pages is a random one, so you might consider using a custom URL.

Access to source code

Both GitLab and GitHub allow for extensive options for visibility of the source code, both private, public or internally. On GitHub, public allows for collaboration with anyone. For SSO login, your book should be part of the GitHub Enterprise of TU Delft. If you'd like a private repo on GitHub, apply for a GitHub Team (free as a teacher as described in the github documentation) to make use of GitHub pages (for public repos GitHub pages is not restricted).

The TU Delft GitLab requires SSO login for editing the book. Although this is useful for TU Delft employees, it limits the collaboration with people outside of TU Delft.

Book size limits

In GitLab, an artifact (the book export) can have a maximum size of 150 MB. In GitHub, the total GitHub pages may not exceed 1 GB (including all branches)

Integration with GitHub Desktop

GitHub has a nice integration with the GitHub Desktop application. For GitLab it works as well, but has less functionality.

Intergration with Utteranc.es

Utteranc.es requires a GitHub repository to host the discussions. If you're using a GitLab repository, you need a separate GitHub repository and the discussions and book content is not at the same place.

Summary

Here's a table summarizing the information:

	GitHub	**TU Delft GitLab**
Book url	GitHub pages (`<organization/username>.github.io/<book>`), for TU Delft books a custom URL (`oit.tudelft.nl/<book>`), for private books on TU Delft GitHub Enterprise with SSO a random URL (`<random>.github.io/<book>`), or custom url `<anything>.<anything>/<book>` > 🌐	TU Delft OIT (`interactivetextbooks.tudelft.nl/<book>`) 🎓
Real-time book editing	Automated and flexible (multiple version of the book, building error insights, fast, custom urls) 🚀	Automated but simplistic (not easily adaptable, no caching environments, no visual summaries, no parallel processes) 🚜 For TU Delft OIT: restricted adaptations because of copyright checks 🚫
Setting up book website	Immediate and automated with template ⚡	Manual setup on personal webserver, or access required by TeachBooks or TU Delft OIT 🚧
Book access with SSO	Only available for GitHub pages on GitHub Enterprise of TU Delft 🎓, optional with custom URL ✅	Optional ✅
Access to source code	Private (if part of organization linked to educational account) /public / internally TU Delft (on GitHub Enterprise of TU Delft) 👥	Private / public (read-only) / internally TU Delft, editing requires requires SSO login 👥 👀
Book size limits	1 GB for all branches 🗂️	150 MB per book 📕
GitHub Desktop	Well integrated 😎	Basic integration 🙂
Utteranc.es	Can be linked to same repository 🏷️	Requires GitHub repository next to GitLab repository 🏷️ 🏷️

If you have doubt about this choice, we advise you to start on GitHub. Moving/duplicating your content to GitLab or hosting to a custom URL is always possible at a later stage.

Install & authenticate required software

Now that the types of users have been explained, you might have an idea which type corresponds to you!

The following table will link the relevant installation and setup steps for you.

User Types	Required actions	Installation
Type 1 Type 2	Review book	No software needed! Just open your interactive book online and give a review in the way you're asked to
Type 3	Edit single files online	No software needed! But you'll interact with Git, which might be a new thing for you
Type 4	Editing text Git	VS Code GitHub Desktop / Git in VS Code
Type 5	Editing text Git Managing code-related software Build your book	VS Code GitHub Desktop / Git in VS Code Anaconda Jupyter Book

Editing text: VS Code

Contents

VS Code is a text editor with some extra features, such as integrated source control and a file browser. Moreover, extensions are available to add functionality to VS Code.

Installation

VS Code can be downloaded from here. The installation is fairly simple and won't be covered here.

Extensions

When working with Jupyter Book (or Notebooks in general), these extensions are highly recommended:

- User type 4-5:
  - MyST-Markdown - adds support for MyST markdown, including previews.
  - Spell checker - Spelling checker for source code and text
- User type 5
  - YAML - enables validation of `.yml` files (the format of the Jupyter Book configuration files).
  - Python - for obvious reasons. Also includes Jupyter Notebook rendering.
  - GitHub Copilot - Your AI pair progammer
  - Jupyter Cell Tags - Jupyter Cell Tags support

Git: GitHub Desktop / Git in VS Code

Contents

In this section you will learn how to install Git locally. Courses on Git usually make use of the command line interface to do run Git, but this introduces a steep learning curve. Instead, using Git from VS Code / GitHub Desktop is much more visual (i.e., more "clicky"), and therefore easier to use for those inexperienced with the command line.

Installing Git

First, we need to install Git itself. Although this isn't strictly required when using GitHub Desktop

git on Windows | **git on MacOS**

1. Go to this webpage and download the correct version of Git for your system.
2. Execute the installer and go through it. You can leave all options to the default values, **but make sure the option "Git from the command line and also from 3rd-party software" is selected**. Without this option, we won't be able to use Git from VS Code.
3. Once the installation is finished, check if everything works. Open the start menu, then type `git`. From the search results, open the application `Git Bash`. This will open a terminal window, in which you can execute commands. Type in the command `git`, and execute it by pressing enter. If Git is installed correctly, you should see some sort of help page. If it outputs an error, then something may went wrong with the installation (you may have selected a wrong option).

Setting up SSH-keys

When we want to visit repositories on GitLab or GitHub, we need to log in. The same holds if we want to interact with these repositories with Git from VS Code. This means that everytime we make a commit, pull, et cetera, we need to provide our password, which becomes very tedious. Alternatively, we can provide GitLab/GitHub with an SSH-key of our system, so that GitLab/GitHub knows it is indeed us that are making the changes to the repository, removing the need to provide a password. If you're only using GitHub and GitHub desktop, SSH-keys aren't strictly required, but it's still advised you set these up.

1. Open up a terminal window (Git Bash on Windows). Next, type (or copy and paste) the following command:

   ```
   ssh-keygen -t ed25519 -C "GitLab"
   ```

2. You will get output that looks something like this:

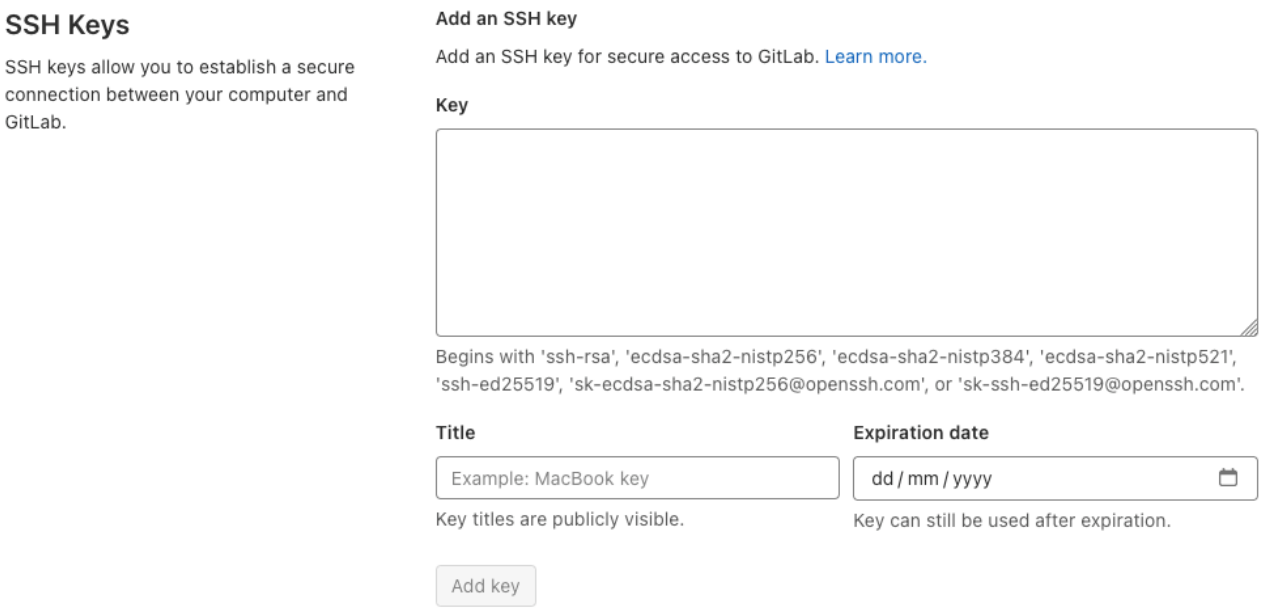
```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/<user>/.ssh/id_ed25519):
```

Press enter to use the default path (the one between parenthesis). Keep note of this path, as we will need to visit it later on.

3. You will now be asked to enter a passphrase. You can give one, but you will be asked for it everytime you make a commit, so it's better to leave this field emtpy and use this key only for GitLab. After the command is done executing, it will generate two files: `id_ed25519` and `id_ed25519.pub`. The `.pub` file is the public part of the SSH-Key which we need.

continue on GitLab | **continue on GitHub**

4. Go to GitLab and log in using your NetID. Click on your profile icon on the top-right of the page, then go to Preferences. In the menu on the left, go to SSH-Keys. You are now greeted with a screen that looks like this:



5. Navigate to the `id_ed25519.pub` file that we generated in step 2. Open the file with a text editor (Notepad on Windows, textEdit on Mac). Copy its contents and paste it in the 'Key' field of SSH-Keys menu. Give the key a Title and click 'Add key'.

> **Note**
>
> On MacOS, the `.ssh/` folder is hidden by default, so you won't be able to find it in finder. To show hidden files and folders, press **Command + Shift + .**

6. Now test the connection. Go to the terminal (Git Bash on windows) and type:

```
ssh -vT git@gitlab.tudelft.nl
```

You should get a huge output, but somewhere at the end of that output it should say something like:

```
Welcome to GitLab, <netid>!
```

Updating the local Git environment

The final step is to provide a username and email address to Git, so our commits can be identified. To do this, in the terminal (Git Bash on Windows) type in the following command for the username:

```
git config --global user.name "<first name> <last name>"
```

And type the following command to set the email address:

```
git config --global user.email "<your TU Delft email>"
```

GitHub Desktop

GitHub is a competitor company to GitLab. It provides very similar services, but they are often called different names, or have slightly different features. GitHub provides a free software that is very useful: GitHub Desktop! While GitLab is a cloud-based version control system built around Git, GitHub Desktop makes it possible to manage our Git repositories locally on our computers, even though they are stored on GitLab. This is very useful for working with the jupyter-book more extensively and work on interactive features. GitHub Desktop provides an alternative to Git in VS Code, and will be described on a later stage.

Terminology

This is a (non-exhaustive) list of terminology that will be used in Git for local use.

- Repository: a collection of files and folders, along with a history of their changes and who made them.
- Commit: a snapshot of the current state of the repository.
- Staging: preparation of files to be committed. During staging, we propose files to be committed.
- Branch: a development line.
- Local: on your own computer.
- Remote: on someone elses computer.
- Pushing: uploading new commits to the remote repository.
- Pulling: downloading new commits from the remote repository to your local repository.
- Tracked (files): files that Git knows about and keeps track of their history.
- Untracked (files): files that Git does not yet know about.

More on GitLab/GitHub

To get to know more about working in GitLab/GitHub and getting familiar with its features, click here or watch this video!

Managing code-related software: Anaconda

Contents

Before you can start using python in your book, you need Anaconda. 'Anaconda's conda tool simplifies package and environment management across operating systems' (Anaconda.com), such as windows or MacOS. It makes getting started with python a lot easier because it already has a lot of useful python packages pre-installed. Unfortunately, each of these packages are themselves require a number of different packages to function properly. Do you like installing all of them? You shouldn't—but where do you get them come from? Many of those packages are packages required by only a few specific packages. These packages are called *dependencies*, and are necessary to make your Python packages function as expected. When you run install a specific package, `conda` (part of Anaconda) checks all of the dependent packages that are needed and makes sure they are also provided in the `environment` (collection of software with a specific version) that is being created. In reality, this is simply a folder on your computer with all of the `*.py` files stored in it. This *package management* is what `conda` and `pip` are really doing when you use them to install a package. It also checks that it has a suitable *version* of each dependency; this is why it sometime takes a long time to install a package Unfortunately, this means that as you add more packages to a particular environment, it gets more and more difficult to make sure everything works well together. Luckily, there is a practical solution: create new environments for specific projects to make sure the proper packages can function properly!

Installing anaconda

You can download the installation files here.

When installing Anaconda, keep in mind the following:

- In general you should use the default file location (don't put it on a separate disk drive, or in a disk partition that is different than your primary OS); however, an exception to this is if you have a space in your Windows username (e.g., `C:/Users/First Last/Program Files`). In this case, install directly ìn a folder on `C:/`
- install it only for your User on the computer (this is especially important on Windows; do not install for all users)

Once installed, we will check to make sure you are ready to use Python. Anaconda provides environments, like small virtual mini-computers inside your real computer, and inside each of those environments you have Python (as well as a lot of Python packages)!

1. Open a command line interface on your computer:
   Windows users: this will be the anaconda or conda prompt (search for it in the Start menu at the bottom left of your screen)
   Mac users: use the "terminal" application
2. Once the command line interface is open, type and execute (hit enter) the following: `python --version`
3. You should be running at least Python 3.11. If not, don't worry, we will try to fix it below (but first make sure you have updated Anaconda as described above)

4. Which environment are you in? The name is between parenthesis at the bottom of your Anaconda Prompt: by default, this should be `(base)`.
5. How many other environments do you have? Execute `conda env list` to see a complete list.
6. Do you see the `*` in the list of environments? That is indicating your current active environment. Unless you have changed something, it should be `base`. And if you just installed Anaconda for the first time, this will be your only environment!

Software version control: environments

A python virtual environment is, like the name might suggest, a reserved space within the computer that contains downloaded python packages. Each environment is like a clean slate and the packages and dependencies of one environment have no influence on other environments. In that sense, it makes sense to create a new (working) environment every time you start a new project. You'll redo the next steps on this page therefor many times in the future!

Set-up environments

Now we will set up your first environment. The following steps will create an Anaconda environment and install Python 3.11. Even if you already have Python 3.11, it is still good practice to create a dedicated Anaconda environment for each of your major projects such as creating a jupyter book.

For simplicity we will call this environment `my_new_book_env`, but you can give it any name you like.

1. Open the commmand line interface (see above) or continue in the same session if it is still active
2. Execute: `conda create --name my_new_book_env pip` (this may take several minutes)
3. Activate: `conda activate my_new_book_env`
   Check: you should now see your new environment displayed somewhere in the prompt between parenthesis, like this: `(my_new_book_env)`
4. Additional 4th step: `pip install -r requirements.txt` (explanation in the following section)

Now everytime you want to work on your jupyter-book, you need to activate the environment! (step 3) If you don't activate your environment, you will work in a 'wrong' working environment that might not contain all the packages you need.

As mentioned before, python packages can be downloaded in the environments you create. Both NumPy and pandas are popular Python packages used for data manipulation and analysis. For the creation of Jupyter Books, we will use the package `teachbooks` which is a wrapper around JupyterBooks. More information about this package here.

Import environments

Working on larger projects (with many cool interactive features) may require you to install a lot of python packages. It can be useful to specify the required packages in a text-based file (for example `environment.yml`) and then telling `conda` to create the environment based on the contents of the file!

Moreover, if you are joining a team that is working on complex projects, it can be useful for you to create a new environment based on use such a text-based file in order to create an up-to-date environment that will give you a flying start. The team might also provide a `requirements.txt` file which specifies all the packages you need to download in order to work on the project. Now it should also become clear why its preferable to create new environment for each project/book that you work on because they might require you to use different packages!

Here's an example of what a `requirements.txt` file might look like.

```
# first list the packages you wish to download from PyPI
sympy
teachbooks
jupyterbook_patches

# now list the packages (and the respective url) you wish to download from the GitLab package registry
--extra-index-url https://gitlab.tudelft.nl/api/v4/projects/11239/packages/pypi/simple
sphinx-thebe ~= 0.9.9
```

This file needs to be manually updated by the team everytime a new package is required. It will be useful to routinely update the packages in your environment by downloading the required packages again. The packages used by sphinx-thebe or JupyterLite-Sphinx are not influenced by `requirements.txt`.

You can do this by navigating your command line interface to the folder with `requirements.txt`:

Windows | **Mac**

1. Open the folder you wish to use as your working directory in the File Explorer
2. Right-click the folder and click `Git bash here`
3. Confirm you are in the right place by inspecting the path listed in your prompt (it should typically start with be `C:\ ...` and end wit `>`)
4. You can also inspect the contents of the directory by executing `dir`

Now you can install the required packages with: `pip install -r requirements.txt`. The TeachBooks team works with these `requirements.txt` files which makes this an essential step in the set-up of your environment!

Export environment

If you want to create a file listing the required packages yourself, you can do it by creating a file with the following file extension: the `*.yml` file (pronounced "yah-mul"). It is a text-readable file, that stands for "Yet another Markup Language." This is one of *many* types of files that use a particular type of text formatting to give a computer specific instructions. It is very similar to the way Markdown formatting works.

The `environment.yml` file will look something like this:

```
name:  MUDE
dependencies:
  - python=3.11
  - numpy
  - scipy
  - matplotlib
  - statsmodels
  - pip
  - conda-forge::jupyterlab
```

You should see that it uses a colon `:` to list the information, like name and dependencies. This will be processed by `conda` when creating the new environment.

If you want to create an environment based on an existing `*.yml` file, you can follow these steps:

1. You need to have the file locally on your computer (for example by cloning the git repositery that it is in)
2. Open Anaconda Prompt
3. Navigate to your working directory (where this file and `environment.yml` is located)
4. Finally, execute this command: `conda env create -f environment.yml`

This may take several minutes because you are installing many packages at once! Keep an eye on the terminal window as this process is completed. First `conda` is collecting information about the dependencies, then it will *solve* the environment;

in other words, figure out which version of each package it should use. Once it is ready, it will present the list of packages and proceed with the "installation" (really just downloading `*.py` files and putting them in a folder on your computer) Note that the prompt may ask you to confirm that the installation should proceed, depending on your system settings.

Once the environment is created, we can activate it, and also check that everything was installed properly. Try `conda env export -n my_new_book_env` to see what was installed by "default." The list is very long, even though we only asked for a few packages!

It is also interesting to try `conda env export --from-history` (make sure you activated it already), which shows the specific packages requested.

Deleting Environments

Although the environments do not interfere with each other, you might want to delete environment you don't use. Try running the following in a Terminal or Anaconda prompt, which will list the file locations of your environment.

```
conda info --envs
```

If you have many projects, the amount of environments can take up a lot of space on your computer as one environment can get very large. It is a good idea to remove environments once you know they are no longer useful.

To remove an environment, in your Anaconda Prompt, run:

```
conda remove --name my_new_book_env --all
```

You will be asked to confirm the deletion, then it may take a while to remove all of the files. To verify that the environment was removed, in your terminal window or an Anaconda Prompt, run:

```
conda info --envs
```

The environments list that displays should not show the removed environment.

This information has been taken from the Anaconda documentation.

Now you can delete any other useless environments you have lying around on your computer!

Combining Git Bash, VS Code and Conda

Contents

Conda in Git Bash

Git Bash is a shell (also referred to as "the terminal") that is automatically installed with git. It provides Unix-like commands on a Windows OS, and is recommended for working with open source projects like Jupyter (for example `pip install`, building Jupyter books, using Git Bash in VS Code, etc). To do this, `conda` needs to be setup properly; this is described here:

Windows - Git Bash (preferred) | **MacOS** | **Windows - Powershell**

There are two things to do before you can use conda in your Git Bash CLI: 1) adding Anaconda to your PATH variables (as user, *not* as system/admin), and 2) setting up Anaconda to run on Git Bash.

1. update the `PATH` variables:

- search for 'env' in the windows search to find the settings page. Anaconda doesn't recommend adding them to the system `PATH`, therefore when searching for the setting with the Windows search, so be careful to open **"edit environment variables for your account"**; do **not** "edit the system environment variables".
- Identify the box for adding variables for a specific user (the top box), not the system.
- select the line with 'Path' and click edit
- Add the `Anaconda3` and `Anaconda3/Scripts` directories to `PATH`.

You will know if this worked because the `conda` commands below will not produce an error. You can also check by opening a Git Bash terminal and listing your Anaconda environments with `conda env list`.

2. To setup Anaconda, open git bash and run the collowing commands (note, the first two commands to create `myenv` can be skipped if you already have an Anaconda environment; e.g., default is `base`):

- `conda create -n myenv python`
- `conda install -n myenv numpy`
- run `conda init bash` to link conda to Git Bash
- restart shell (type `exit` to close)
- run `conda activate myenv`
- restart shell

Check if it worked by opening a notebook in an aribtrary `DIR` using the command `jupyter lab --notebook-dir DIR`.

Using Git Bash in VS Code (Windows only)

By default, Powershell is the default terminal used in VS Code. However, Git Bash is recommended because it provides Unix-like commands, making it much easier to find help on sites like Stack Exchange. It also works better with open source

projects like Jupyter. Git Bash installs automatically with git which you'll learn in the next chapter; to set as default:

- Type `Ctrk+Shift+P` to open the command palette (it's also the top item in the "View" menu list)
- Start type "terminal default" and you will soon see and be able to select the setting "Terminal: Select Default Profile" (hit `Enter`)
- select Git Bash and hit "Enter" again to set as default

When creating a new terminal via the menu bar you should now automatically start with a Git Bash terminal. Note that if a Powershell terminal still opens by default, you can always create a new terminal and manually select Git Bash.

Build your book: JupyterBook

Contents

jupyter{book} is a regular Python package which converts our content into a website. It's part of all of the `requirements.txt` files for a book repository. At TeachBooks we developed a Python package called `teachbooks` which takes over the these features and adds additional features. One of those additional features, for example, is the workflow to GitHub Pages.

Concretely, this means that the teachbooks package can be used to replace the jupyter{book} package when calling these features. You could think of it as `jupyter-book` being a dependency of `teachbooks`.

> **Note**
>
> Don't forget to add the package `teachbooks` in the `requirements.txt` files in your book repository and to have it installed in your working environment (`pip install teachbooks`).

Build a book

As soon as you've installed your book environment with jupyter-book you can build your book locally. The official jupyter{book} documentation is quite extensive. You can navigate to the directory where your book is to leave out the `path-to-book`.

```
teachbooks build <path-to-book>
```

Alternatively:

```
jupyter-book build <path-to-book> --all
```

Note: use `--all` for the jupyter-book whenever building a book to make sure the table of contents fully updates. Or alternatively, delete the old `_build` folder before building the new book.

View the book locally

Once the build process is completed successfully, you will see the file location in your terminal output. Your terminal will look something like this. Simply open this in the browser (you may have to copy/paste).

```
================================================================================

Finished generating HTML for book.
Your book's HTML pages are here:
    book\_build\html\
You can look at your book by opening this file in a browser:
    book\_build\html\index.html
Or paste this line directly into your browser bar:
    file://C:\Users\Julie Kirsch\Documents\TUDelft\TeachBooks\Q3\jupyter-book-manual\book\_build\html\index.html

================================================================================
```

Fig. 3 Terminal

Tip

Bookmark the (local) file location for easy access!

Collaborative book-editing

A jupyter book is composed out of many files which contain the educational content produced by you. Through the use of branches it is possible to work on the book at the same time as your team-members. Working on content for the book is mainly individual work, the beauty of Git comes in when everybody's changes are merged into the main branch.

We have previously introduced the notions of using Git in the online environment (remote) or locally. All the (text, figures, etc.) files that make up the book are contained on the remote repository on GitLab or GitHub. This could be on an open repository on `gitlab.com` or `github.com`. In the context of creating educational books for students at TU Delft, the repository will most likely be located on `github.com` or `gitlab.tudelft.nl`. If this is the first time you interact with books, you're advise to use `github.com`.

The steps involved are as follows:

1. **Assign a task to yourself (milestones, issues)**
2. **Create your own version of the book as...**

... user type 3 ... | **... user type 4 and 5.**

... by creating your own version of the book (branching) or selecting existing version:

- branching

3. **Edit the book as ...**

... user type 3 ... | **... user type 4 ...** | **... user type 5 ...**

... by directly adding changes on a single file to Git-timeline (committing):

- adding file or making changes to single file
- committing

4. **Check changes online**
5. **Repeat steps 3 and 4 until you're satisfied**
6. **Reviewing (merge request), (eventually repeat steps 3 and 4) and combine (merging) your version with main draft version of book**

Assign task to yourself (milestones, issues)

In a repository, team members can define Milestones and Issues. Milestones are used to track the overall progress of the project to make sure that deadlines are met. A milestone can group many issues, which are individual tasks that need to be finished in order to reach the milestone.

An issue is built up of the following attributes:

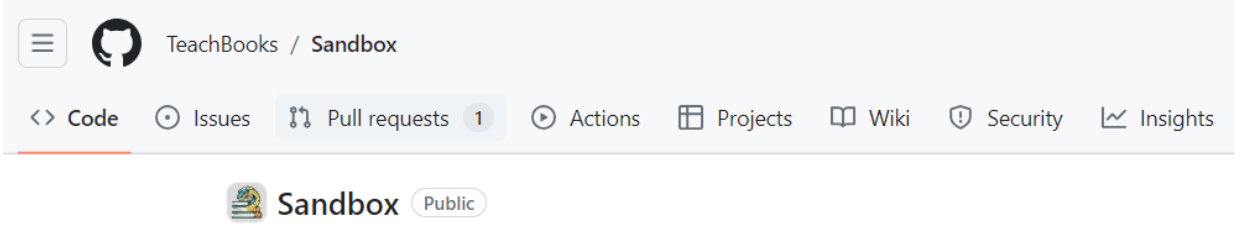
- the title, this should just be a quick description of the task
- a label, to assign a category to the task such as: new content
- an assignee, the person who created the task can assign a team-member to the task or the task can remain without an assignee. Then whoever would like to pick up the task can assign it to themselves!
- a discussion section, here team-members can discuss any questions about the task
- a due date, in case the task is urgent or to keep track of deadlines

Every team member can create issues when they think of a task that would improve the book.

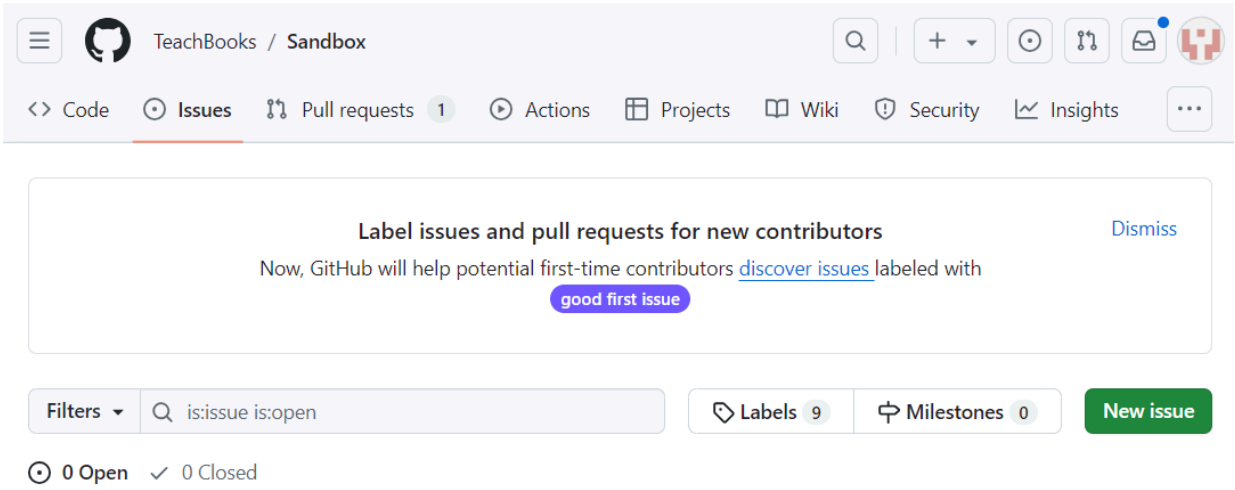
Make and assign task to yourself (issues) in ...

... GitHub | **... GitLab**

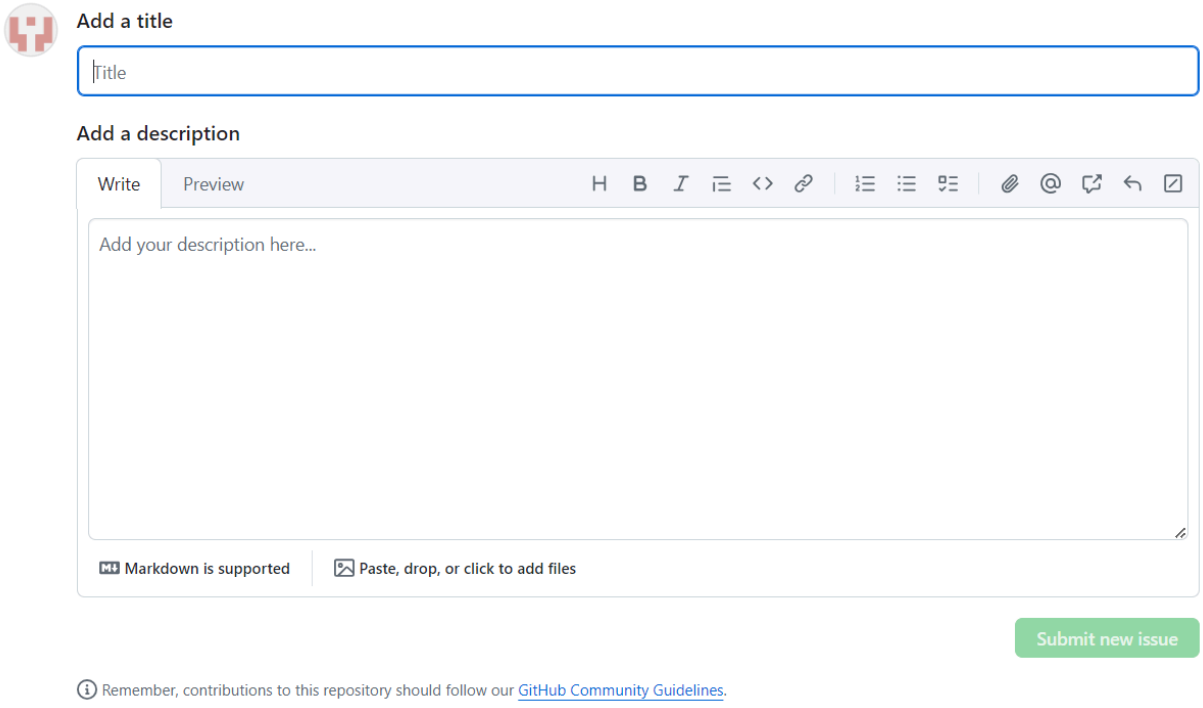
The issues in GitHub can be found in your repositery in the top bar.



Clicking on `Issues` will open up the page with all the issues on this repositery.

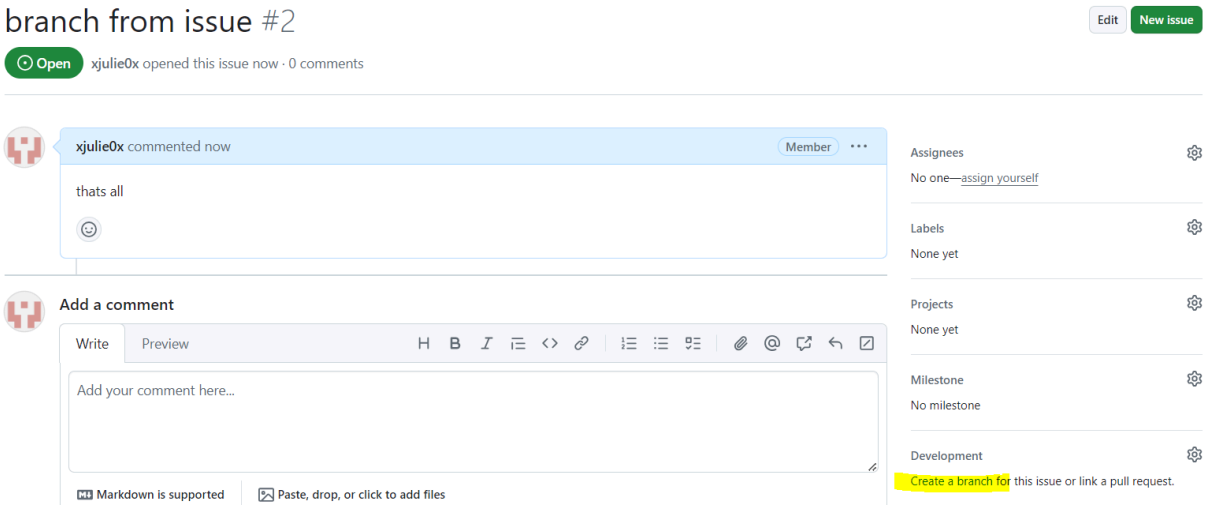


Finally, by clicking the green `New Issue` button you can create a new issue and specify all the attributes we mentioned before!

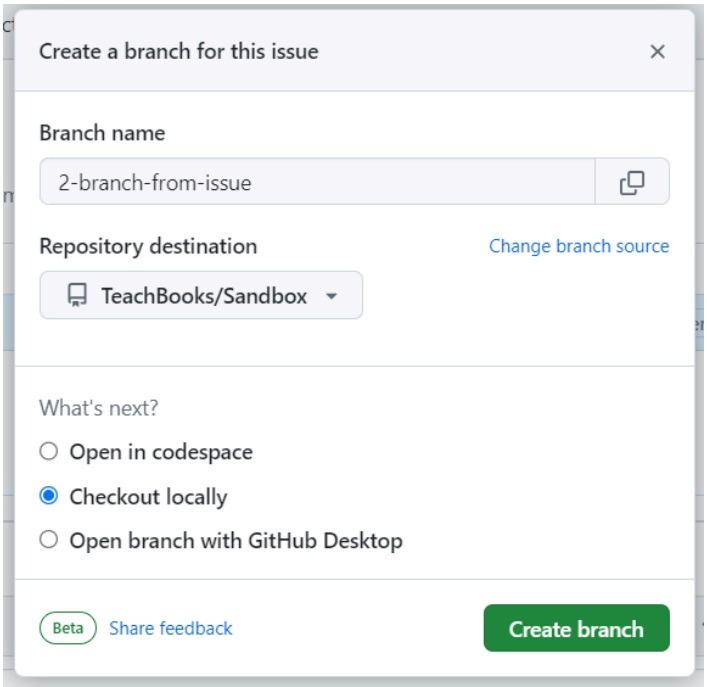


> **Note**
>
> The follwing steps explain how to create a branch from a new issue. By doing this you can create most of the steps described in the section called branching. However, if you are unfamiliar with the term make sure to still read the section to understand the function.

After creating the issue, the description of the issue will automatically open. On the bottom left there is an option `create a branch` (in yellow) from this issue. This will create a new branch in the repositery which you can use to solve the issue.



Clicking on `create a branch` will open the following window in which you can specify the name and the repositery.



When the task is completed, the issue can be closed and you can move on to the next!

Create your own version (branching, cloning, pulling)

Branches are a very useful feature of Git. Branches allow you to work on multiple versions of your codebase simultaneously; you create a copy of your codebase, on which you can work independently of the main codebase. The main branch is the default branch. The content on this branch will be the most up to date version of the book. Therefore, new branches, which are made to add or fix some content in the book, are usually cloned from the main branch. This has many advantages compared to making the edits immediately in the main branch. Some advantages of working with branches are:

- Isolation: when you work on your own branch, your changes are isolated from the main codebase. Unfinished or unreviewed parts of your book are "hidden". Your own branch helps to keep an overview of the changes made to the book.
- Collaboration: branches make it easier for multiple project members to work on the same codebase simultaneously. If every member works on their own branch, they can make their changes without having to worry about interfering with another members' changes. In case those changes made in parallel lead to conflicts, they can be resolved during the merging of the branches.

This systematic workflow guarantees that editing the book goes smoothly.

The act of making a new branch from an existing one is called *branching*. Usually, you want to branch from the `main` branch, but you can of course also choose to branch from another branch. The branch from which you create a new branch is called the *source* branch.

Create your own version as ...

... user type 3 ... | **... user type 4 and 5 ...**

... by directly creating your own version of the book (branching) or selecting an existing version in ...

... GitHub | **... GitLab**

How to branch is demonstrated in the figure below, all steps are elaborated on in the following step-by-step tutorial.

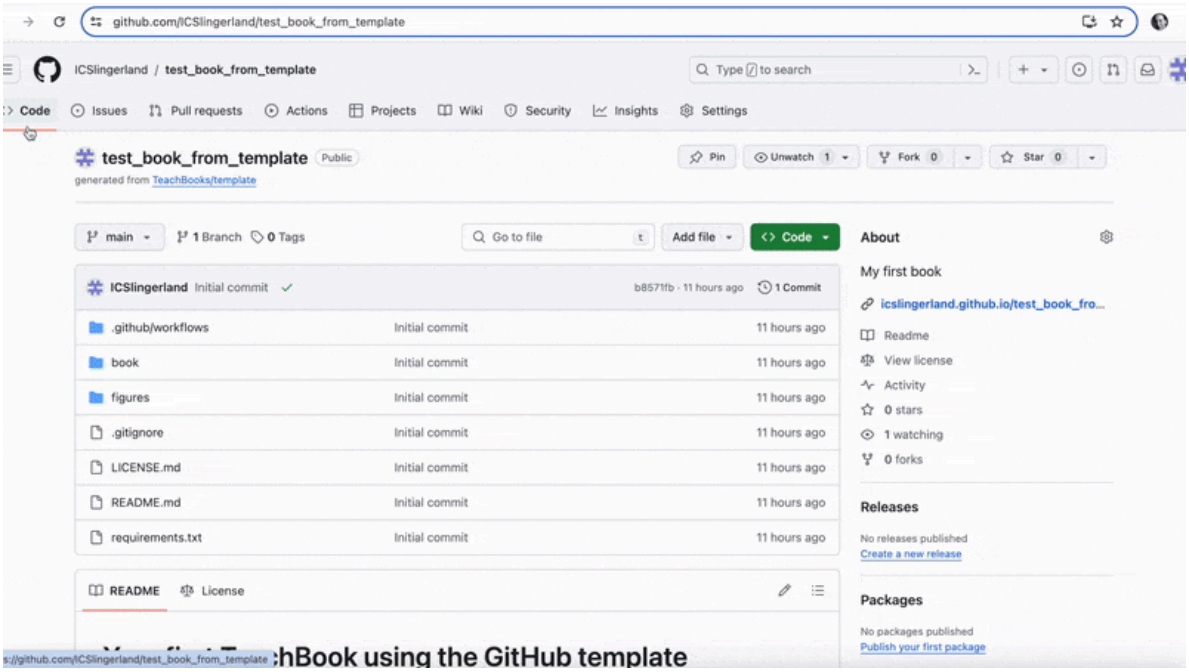


Fig. 4 Demonstration, video available here

1. Open the repository in which you want to work in. In the top bar you will see your current branch `main` and next to that there is a tab called `Branches`.

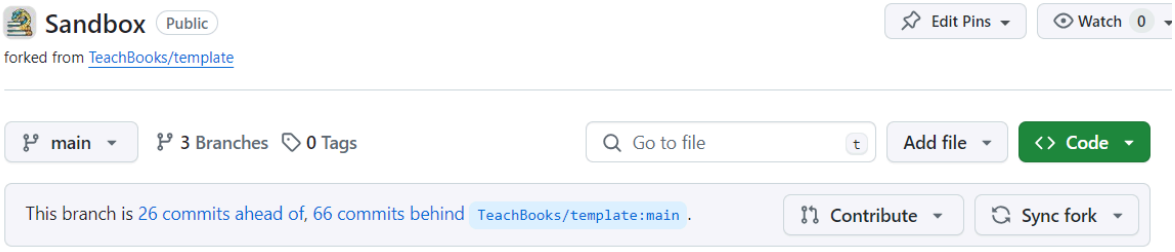


Fig. 5 Overview

2. Clicking on `Branches` will bring you to an overview of all the branches in the repository. Click on the green `Create Branch` button.

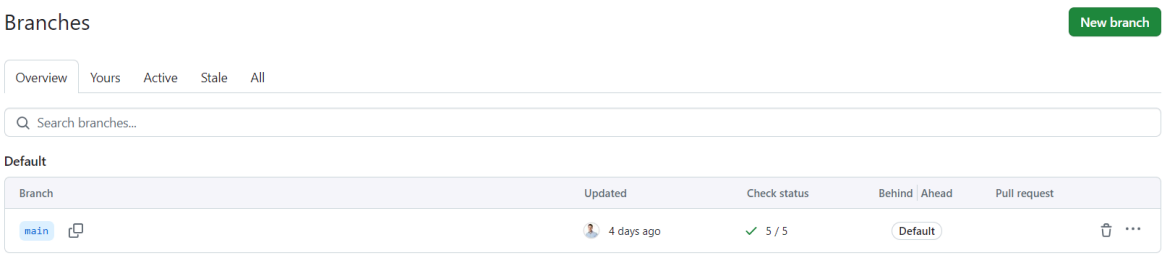


Fig. 6 Overview Branches

3. Finally, give your branch a name and specifiy the source branch.
   Tip: It makes sense to give the branch a name related to the content you will be creating or altering.

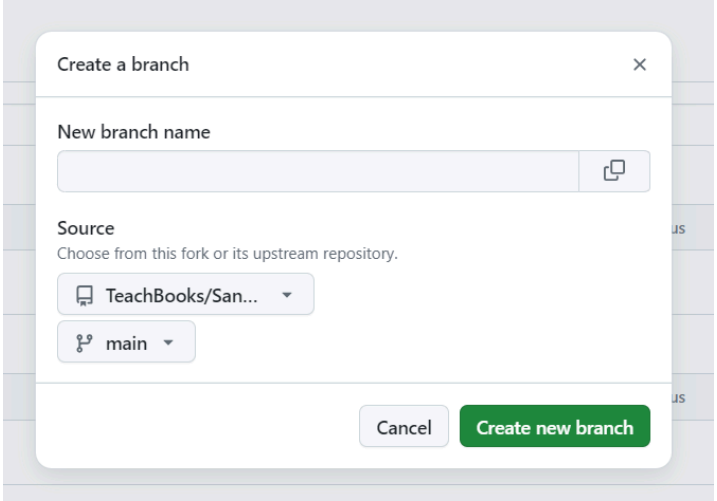


Fig. 7 New Branch

4. You will find your new branch in the drop down menu as seen on the figure below. You can then select your new branch and start working in it (new version) but you can also work in any other branch if you are helping out a colleague with their chapter for example (existing version).

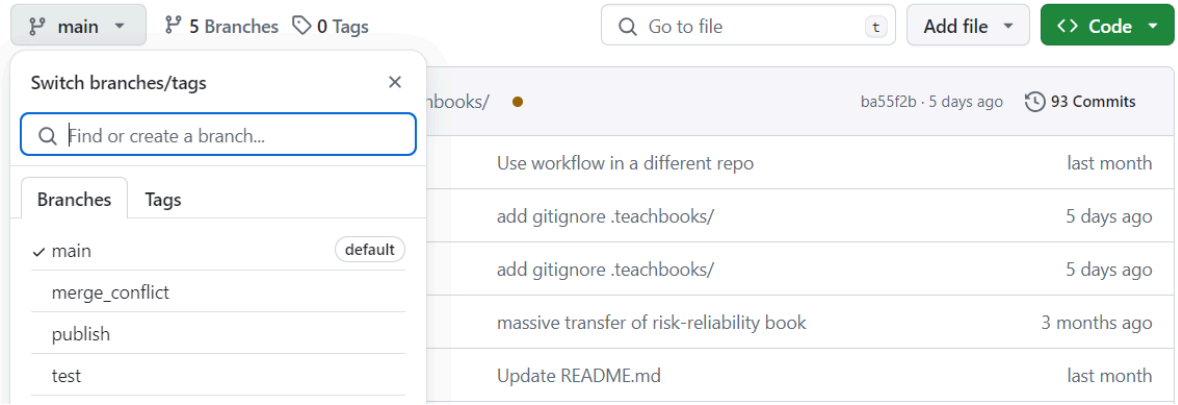


Fig. 8 Select New Branch

Edit (merge conflicts, staging, committing, pushing)

Suppose we are writing a new chapter, or are updating an existing chapter for our Jupyter Book. We've created a new branch on which we are going to make the changes.

Edit files as ...

... user type 3 ... | **... user type 4 ...** | **... user type 5 ...**

... by directly adding changes on a single file to the Git-timeline (committing) in ...

... GitHub... | **... GitLab**

... using 'Edit in place' | **... using 'VS Code in your browser'**

How to make an edit and make a commit is demonstrated in the gif below. Don't worry if the steps are too fast, all steps are elaborated on in the following step-by-step tutorial.

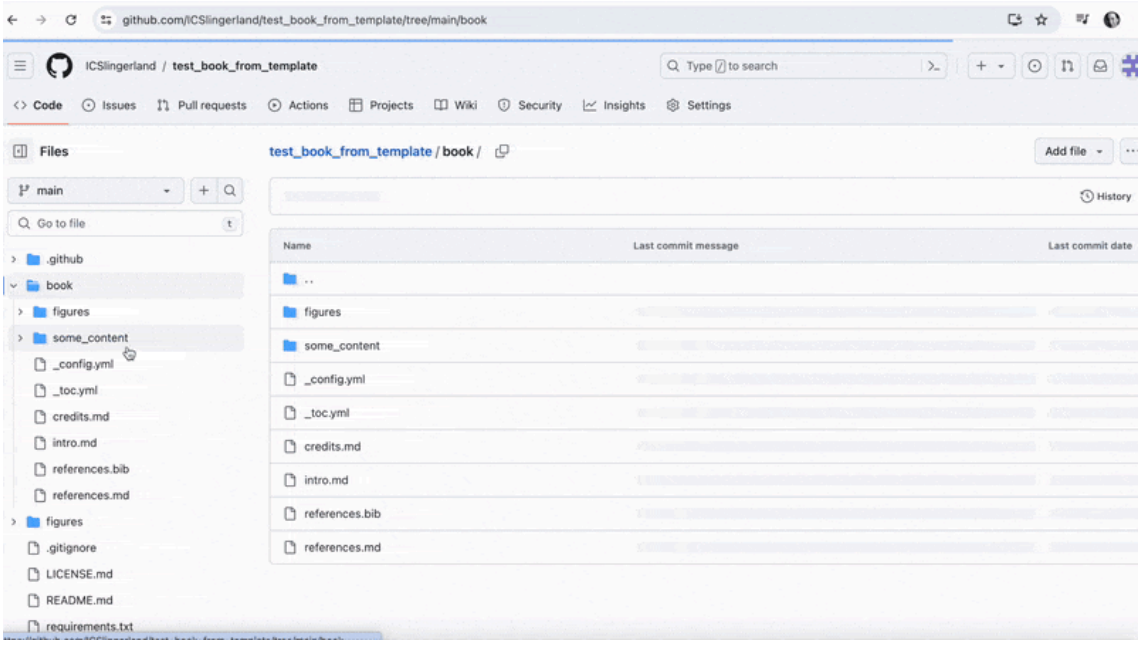


Fig. 32 Demonstration, video available here

In GitHub, you can directly make changes in the files on the remote repository. You can make changes to the files already in the repository using the text editor

but you can also upload new files!

1. Navigate to the repository you want to work in and make sure you're in the correct branch.
2. Create a new file by clicking on the button called `Add file` in the top bar. You can either create an entirely new file by clicking `Create new File` or if you already have created a file you can upload it by clicking `Upload files`.

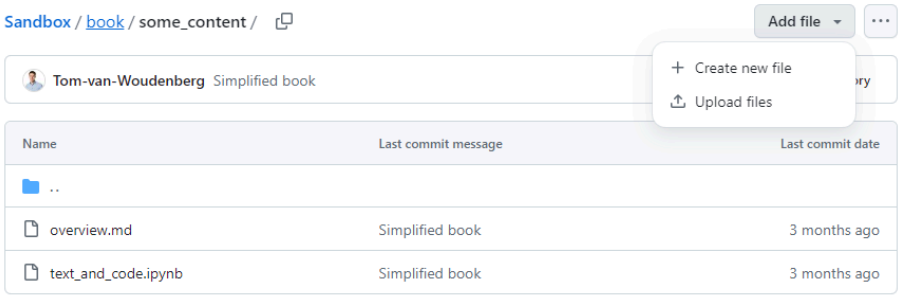


Fig. 33 Create new file

3. In the new window, you can start typing your content. Give your file a name and make sure to use the markdown extension: `file_name.md`. Once you are done, commit the new file to the repository by clicking the green button `Commit changes`.

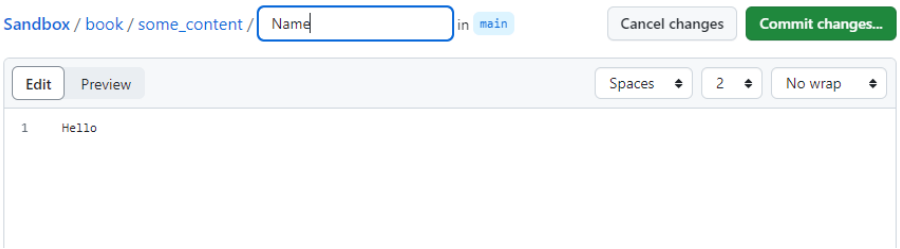


Fig. 34 Edit new file

4. In case you want to make changes to an existing file, navigate to the file in your remote repository. Then click the downward pointing arrow on the very left in the top bar. Select the option `Edit in place`.

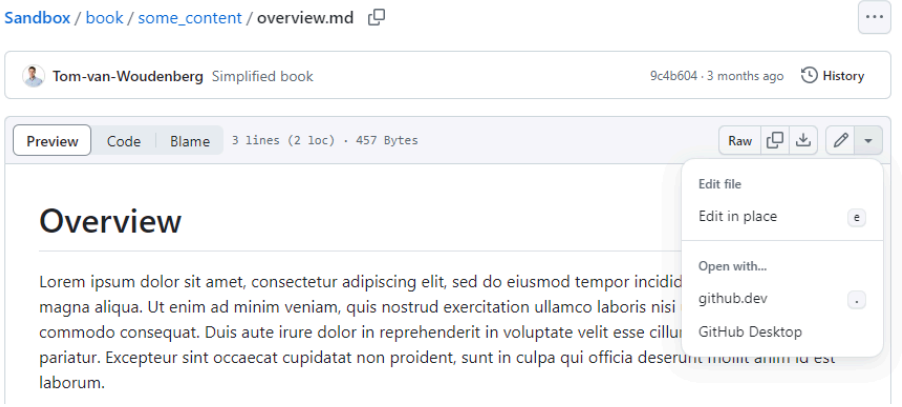


Fig. 35 Edit in place

5. Make your changes in the text editor and when you are ready, commit your changes to the remote repository by clicking on the green `Commit changes` button.

Fig. 36 Commit changes

6. In case you added a new file, you also need to include it table of contents of your book! The table of contents is specified in a file called `toc.yml` and it is already included in your repository if you used the teachbooks template. You can edit it and commit the changes in the same manner as with markdown `(file.md)` files.

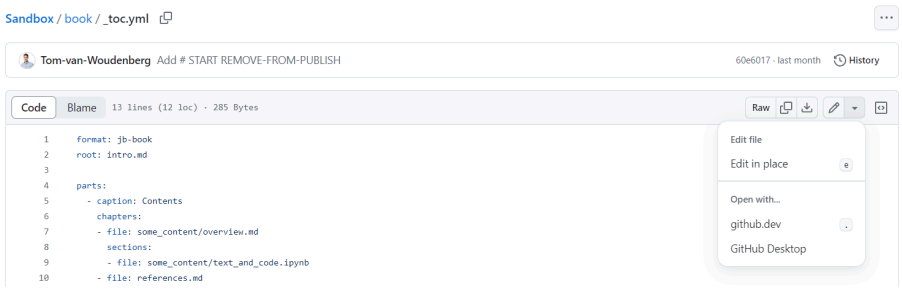


Fig. 37 Table of contents

Check changes online

> **Note**
>
> This page is relevant for user type 4 and 5. When making changes to the book it can be useful to get an idea on how they will be realized in the online book. Whereas user type 5 can also build the book locally, user type 4 is more focused on the changes to the content.
>
> However both users would certainly benefit from checking whether their changes look good in the book!

Now that you've pushed your changes online, it's time to check the change in the online build book to make sure that your changes are processed correctly. For this step it is assumed you're using TeachBook's GitHub actions or TeachBook's GitLab pipeline.

In GitHub | **In GitLab**

Once GitHub Pages is enabled and your book is published, you can view it by visiting the GitHub Pages URL for your repository. A summary can be seen under the `Actions` - `All workflows` - `call-deploy-book` in GitHub. It shows you a descriptive summary. The summary also provides build error messages, which you might fix in a new commit.

If this is not activated yet, you can do so as describe here. Here you can also find more information.

You can keep on editing and checking your changes until you're satisfied and ready for a review by a colleague.

Reviewing and combine versions (merge/pull request, merging)

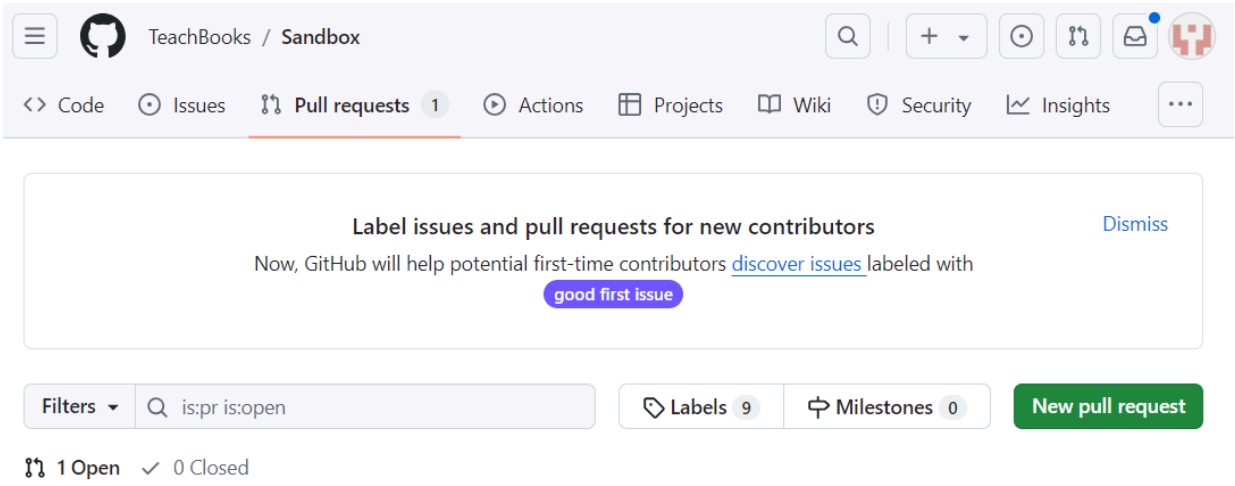
Merging a branch into the main branch is also good moment to let a team-member review the content. By assigning a colleague to review your branch you can check the quality of your new content.

> **Note**
>
> There is a slight difference in terminology between GitLab and GitHub. GitLab uses the term *Merge* Request while GitHub uses the term *Pull* Request. Both actions refer to the merging of a branch into the main branch.

We're finished with our chapter, and now it's time to include all of our commits in the `main` branch. However, it's good practice to first merge `main` into our branch, so that we can see if there are any merge conflicts. Assuming that those merge conflicts are solved before, we'll now continue this example with the following merge of our branch into `main`.

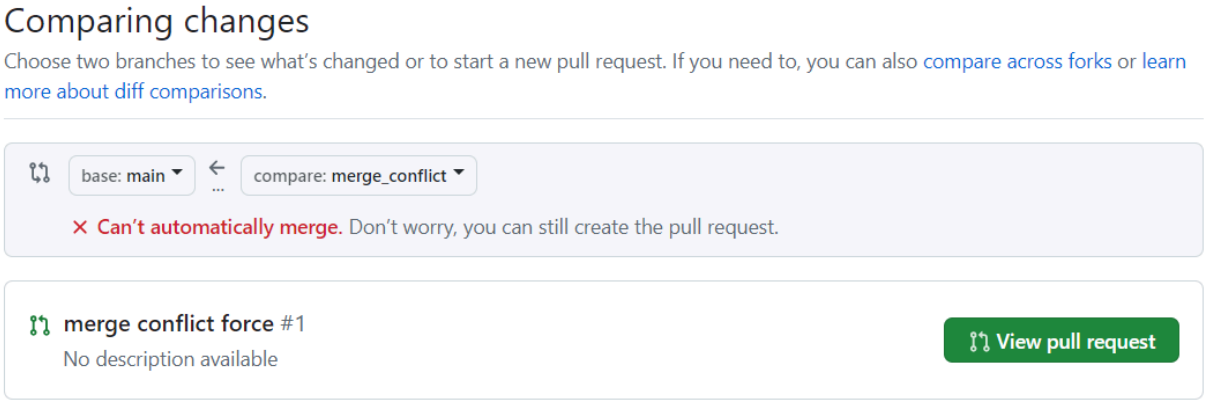
In GitHub | **Locally with GitHub Desktop for GitHub repositories** | **In GitLab**

1. Make sure that you've committed all your new sections and changes, and that they're pushed to the remote repository.
2. In GitHub, navigate to the repositery you want to work in. In the top bar click `Pull Requests`. Then, in the new window click on `New Pull Request`.



3. You can now choose a base branch and a compare branch.

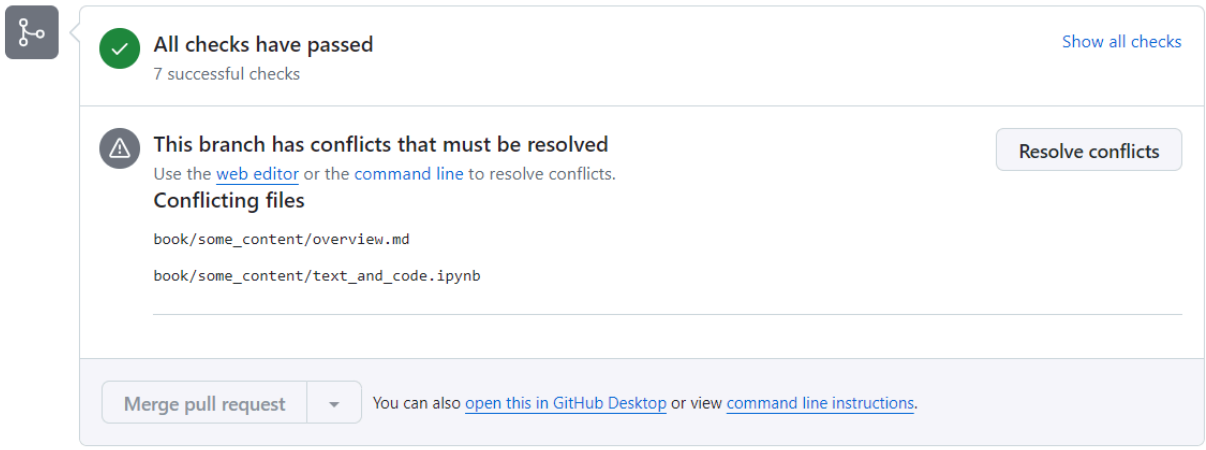
- Base Branch: The base branch is the branch into which you want to merge your changes.
- Compare Branch: The compare branch, also known as the “head” branch, is the branch that contains the changes you want to merge into the base branch.



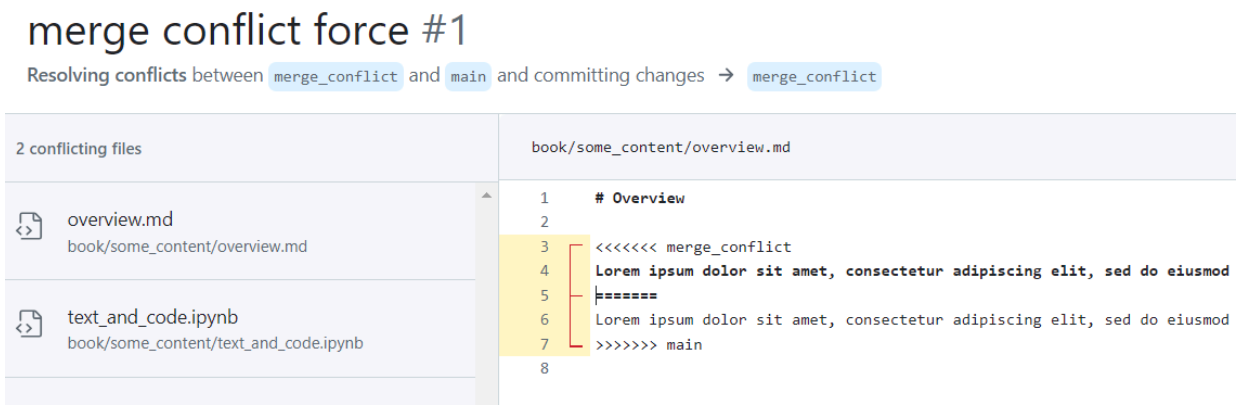
In the figure, you can see that we want to merge our changes from the branch `merge_conflict` *into* `main`, `merge_conflict` is the compare branch and `main` is the base branch. Note that in the figure it says that the branches cannot be automatically merged. The reason for this is that there are conflicts in the two branches, which you can see when scrollong down.

You can still create the pull request but you will have to manage the conflicts. Once you’ve selected the correct branches, click `View Pull Request`.

4. As previously mentioned, before merging the two branches we have to resolve the conflicts. Click on ‘Resolve conflicts`. Note that if your branches do not have conflicting changes you can merge directly and skip this step.



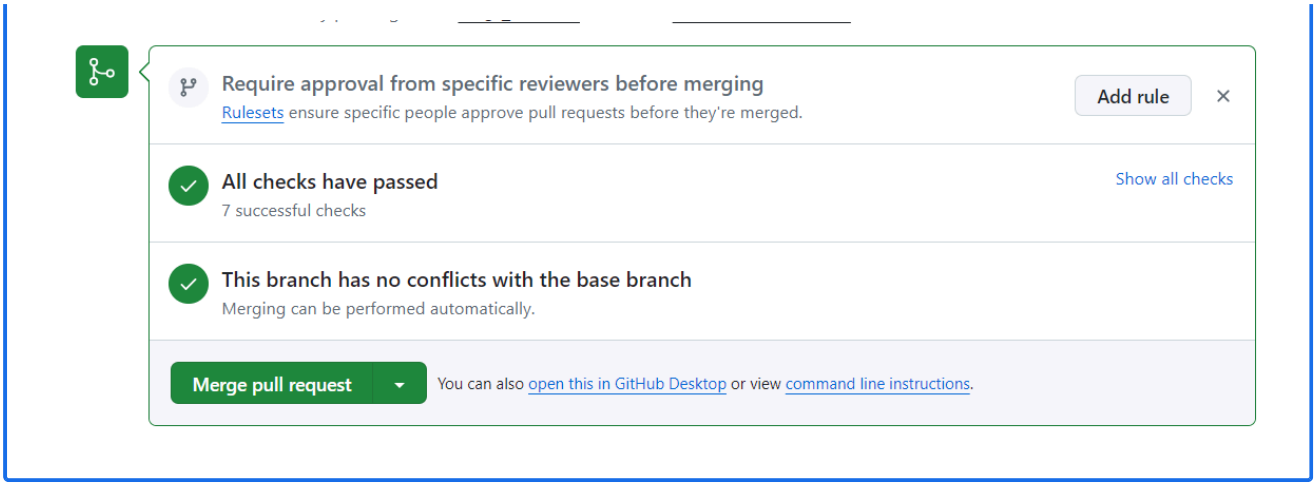
Clicking on 'Resolve conflicts`, will open the conflicted files in the text editor. There you can manually edit the files to the version you would like to retain.



Once you are done, click on `mark as resolved` in the top right of the window. Do this for all conflicted files. In case you have cloned the repositery you can also resolve the conflicts locally using VS Code for example. You can read more about it [here] (edit_book.md#... GitHub Desktop)

5. Commit the changes by clicking on `Commit merge`.

6. You can now complete the pull request by clicking `merge pull request`.


Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged.
Add rule
All checks have passed
7 successful checks
Show all checks
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.

Organize editing team, collaboration and visibility

Contents

Roles and responsibilities

To ensure smooth collaboration, it is essential to define the roles and responsibilities within your team. Firstly, designate who will be in charge of team organization ('administrators') who should be at least user type 4. Additionally, assign 'maintainers' responsible for combining content, also requiring at least user type 4. 'Editors' should be identified for editing and reviewing content, requiring at least user type 3.

Finally, ensure that team members identify themselves according to the user types. It is crucial to have at least one user type 4 or 5 on the team. While this role can be filled by a teaching assistant (TA), it is recommended that one of the teachers is also comfortable with this user type to provide adequate support and oversight.

Review process

Furthermore, establish a clear editing and review process.

For the editing process, establish some practical rules on whether to use issues, branches and forks for adding content, fixing typos or making other changes. Furthermore, it can help to write down when you expect (draft) merge/pull requests and how to use the assign and review options in GitHub/GitLab. It is advisable to have at least a maintainer review each piece of content to maintain quality and consistency. Provide explanation on how you organize the book-editing and how you'd like to receive feedback in both the readme as the published book. The use of the repository button, suggest edit and issue button is advised.

Define draft version of book

If there's many people working on different parts of the book and material is staged for publication to students, we recommend using at least two separate branches, one which is released to students, the other one for development. Using the Releasing book online, it's very easy to share multiple versions of your book on the same root-URL.

Firstly, create a release branch which contains the students' version of this year. This branch could be named `release`, `main`, or `<current academic year>`. It is recommended to set this branch as the `PRIMARY BRANCH` when using the GitHub workflow. This ensures that students are redirected to a consistent URL, even when new versions of the book are added

later. If you plan on maintaining only one public version, it is advisable to set `BEHAVIOR_PRIMARY` from the default `redirect` to `copy`. This ensures that the primary branch is copied to the root, making the URL more compact.

Next, establish a development branch that contains all the new content combined but not yet published to students. This branch could be named `dev`, `main`, `<current academic year>-draft`. Using `dev` is recommended if you prefer the default branch to be the most recent published version, as `main` is commonly expected to be the default branch.

We advise you to enable two options in the general repository setting regarding pull requests in GitHub:

- Enable `Always suggest updating pull request branches`, suggesting a merge from the default branch into any separate branch before merging into the default branch.
- Enable `Automatically delete head branches` to delete branches after they are merged (you'll still be able to restore those).

Book previews of contributions

Using the Releasing book online, it's very easy to show a preview of the changes for branches within the same repository. Using Build pull requests from forks you allow previews of changes for forks of your repository too.

Add the way you organized branches and builds in your README.

Protect branches

To ensure the integrity of your book's content, it is crucial to set up protected branches in both GitHub and GitLab. Protected branches prevent accidental changes and ensure that only authorized modifications are made. In GitHub, you can configure branch protection rules by navigating to the repository settings, selecting "Branches," and then adding branch protection rules for the branches you want to protect. Similarly, in GitLab, you can protect branches by going to the repository settings, selecting "Repository," and then configuring the branch protection settings.

Once the branches are protected, it is advised to assign appropriate permissions to team members based on their roles. In GitHub, you can manage permissions by navigating to the repository settings, selecting "Manage access," and then inviting collaborators with specific roles such as "Admin," "Write," or "Read." In GitLab, permissions can be managed by going to the project settings, selecting "Members," and then adding users with roles like "Maintainer," "Developer," or "Reporter."

Additionally, it is advisable to document the branch protection strategy and the assigned permissions in the readme file. This documentation should explain the rationale behind the branch protection rules and provide clear instructions on how team members can request access or permission changes. By doing so, you create a transparent and organized workflow that facilitates collaboration and minimizes the risk of unauthorized changes. We would advise a ruleset:

- Bypassable by repository admins
- Targeting default branch
- Restrict deletions
- Require a pull request before merging with 1 required approval
- Block force pushes

Private, internally or public

Choose whether you want to share your source code and built book privately, internally or publicly. In any case, you need to obey copyright-rules. We advise public repositories and books in line with a vision on open education.

Versioning and URLs

Online books can be easily updated, however, this might confuse readers. Therefore, it's good practice to be aware of this and, where needed, provide an explicit version to the readers.

For example when making taylor-made books for specific academic years, you want students to be able to find their own book (even after the academic year is over). For students which take the course twice, you might want to provide some sort of changelog of what has been changed.

Another example could be a book in which you add content and solutions during the course or you're fixing mistakes which may have misinformed the reader. If this is not directly visible in the table of contents, students might not be aware of these changes. Again, a changelog could be useful to inform the reader about the change. Furthermore, a smart versioning system might tell the reader something about the impact of the addition differentiating errata and additions.

You could also think of your published book which is referenced by other people. Don't be afraid, git makes it very easy to make this possible. Git allows you to go back in the git history. To make this as easy as possible for readers, you can assign versions using TeachBooks versioning instead of having the readers scroll through a long list of (not alway very descriptive) commit descriptions.

Maybe this all sounds confusing and difficult. No worries! A use-case could be a book which progresses in time, but in which you don't want to deal with the additional hassle of versioning. Your book could be perfectly fine without any measures on this topic!

If you are part of the previous group, but at some point decide to make a new version of the book while keeping the old one, you could do so without dealing with version numbers and changelogs. Just publish both versions online.

Finally, you might consider publishing your book at an online publisher. This could increase findability of your book and you might benefit from the brand of the publisher, but it may have flexibility, copyright and licensing consequences.

So, there's a few use-cases for which different (combinations) of solutions exist. These are summarized in the table below in order of increasingly effort and impact. You can combine use-cases to your liking:

Use-case	Publish versions on separate URLs	Use TeachBooks versioning with a changelog	Publish your book with a publisher
Book changes over time, but you don't want to deal with versioning			
Occasionally releasing a new version while keeping the previous version, with minimal effort for versioning	Recommended		
Continuously adding content and solutions or updating crucial mistakes		Recommended	
Allow referencing specific versions	Can be considered	Recommended	
Taylor-made book per academic year	Recommended	Recommended in case of students retaking a course	
Increase findability or benefit from publisher's brand			Recommended

Read more about each of these measures on the following sub-pages.

Publish version on separate (fixed) URLs

As soon as you have different versions online, you have to make a decision about URLs. Using the Releasing book online, you can release the different version of your book all under the same root URL. So, it uses the same workflow as used for draft versions of your book as describe before, the only difference being that URLs of draft versions are generally not shared publicly.

If you plan on maintaining only one public version, it is advisable to set `BEHAVIOR_PRIMARY` from the default `redirect` to `copy`. This ensures that the primary branch is copied to the root, making the URL more compact.

To share the built book / website of old versions for your book on GitHub you have a few options:

- Create a branch for each version. When doing this, the GitHub workflow will build it on GitHub pages. Here's a few tips:
  - It's advised to use the version as branch name. In case of versions per academic year, you can just name each branch to the current academic year, this leads to the URL with the same academic year..
  - It is recommended to update the primary branch as the `PRIMARY BRANCH` when using the GitHub workflow whenever a new version is considered to be the most primary one. This ensures that readers are redirected to a consistent URL, even when new versions of the book are added later.
  - Eventually, you can add old branches to the list of `BRANCHES_ARCHIVED` when using the GitHub workflow to include a banner on the page indicating its archived state. Alternatively, you can add a banner manually to `_config.yml`, as explained here.
  - If you're using tags (as will be explained in later), you can create a branch from a tag locally by running: `git branch <new_branch name> <tag/version>`.
  - If you have many version of the book, at some point you might reach the 1 GB GitHub Pages limit. This is only expected for extremely large books with a lot of non-text-based (binary) content. To prevent going over that limit, store large content not used for building the website (like images) on an external server. This can be another GitHub repository or a (paid) object store.

    - An example can be found in the book 'Engineering Systems Optimization': it has two versions for readers online, the 2023-version and the 2024-version, recognizable by the URL. Both versions originate from their respective branches in the GitHub-repository
- Add a zip of the built book to the release in GitHub as binary asset. This requires readers to download the zip and open the html files locally, but doesn't require you to host it somewhere. This zip can be easily downloaded as an artifact from the GitHub action. An example can be found here. Note that this requires you to set up tags and releases, as explained in TeachBooks versioning with changelog
- Use Read the Docs to build a website for all your branches, forks and tags as Read the docs doesn't have a 1 GB limit, but has a very different URL structure and the free community version has advertisement. In case you're using tags and releases, refer to the URL of this built from your GitHub release to make sure that GitHub visitors find their way to Read the Docs.

If you have your own webserver, you can publish each version on it manually.

Don't forget to explain how you organize your URLs and old versions in your README and eventually in the book itself too with a sentence like this (as can be seen in this book too):

> This is the `<version_number>`-version of this book. Go to `<link to root URL>` to view the most recent version of this book, or adapt the year in `<link to root URL>/<year>` to the year when you took the course.

TeachBooks versioning with changelog

Contents

Versions combined with a changelog can be a very effective way to communicate book changes to the reader and allow reuse of specific versions. We recommend TeachBooks versioning, which comes in two flavors:

1. `academic_year.additions.errata` versioning for books tailed-made for courses in which content is added / adapted during the course and might be restructured extensively every year while remaining to be available in the original form. An example can be seen in the source repository of the Engineering Systems Optimization book showing tags for different versions.
2. `major.errata` versioning for books which are more stable over years, in which big changes are covered only by the version number.

The details of TeachBooks versioning are covered here

Changelog

To communicate changes, we advise creating a changelog in the book. The template contains an empty changelog to fill:

```
# Changelog

## `<latest version>`: `<date>`
- `<Added/modified/deleted>` [](`<relative link to changed file>`)
- ...
- Full Changelog: `[<previous version>...<current version>](<link to diff as provided by (

## `<previous version>`: <...>
- <...>

<...>
```

An example can be found here

Note on version change on page

When making errata changes or additions, it's advised to notify the reader not only in the changelog, but also on the relevant pages. You can do so using the `versionadded` and `versionchanged` admonitions:

```
::::::{versionchanged} <version_number> <date of version release>
<explanation of change on current page>
::::::
```

and

```
::::::{versionadded} <version_number> <date of version release>
<explanation of addition on/of current page>
::::::
```

Leads to for example:

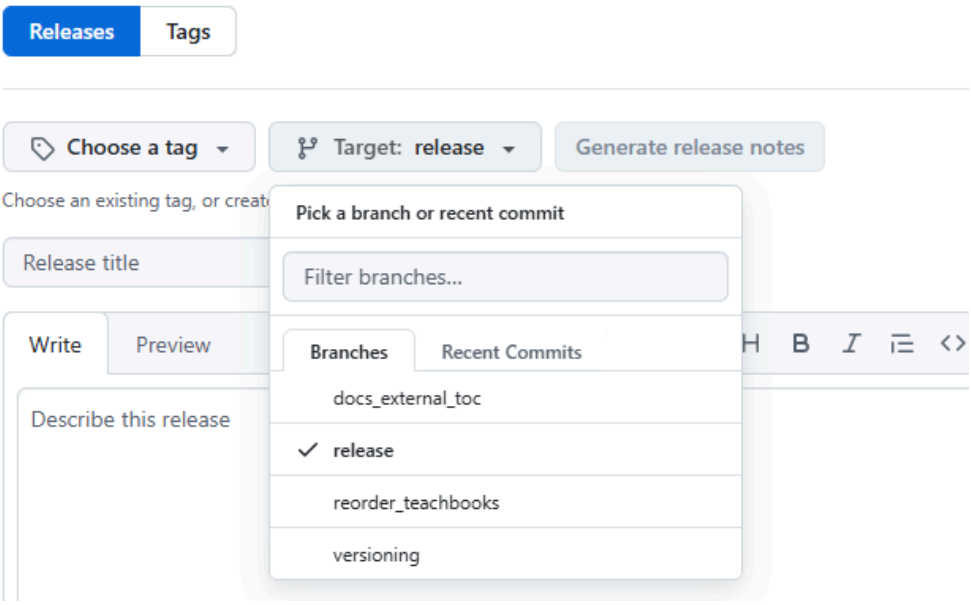
> ***Changed in version v2025.3.5:*** 2025-02-23
>
> TeachBooks versioning added

An example can be found here

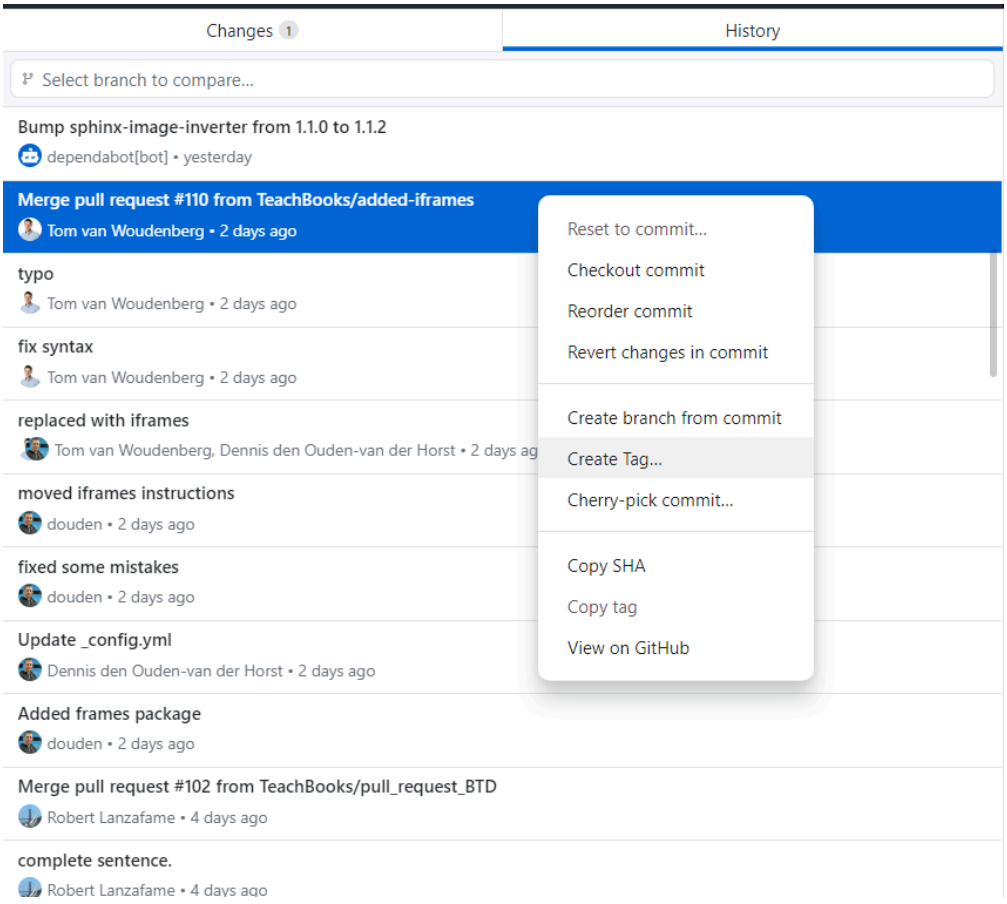
Implement tags and releases

Tags can be added to your source code by adding the version number as a tag to a specific commit.

You can do so in GitHub when creating a new release (on the `Code` page of your repository.) There, you can enter a name for your tag and select one of your branches or recent commits:



Locally, you can do so in GitHub Desktop by clicking 'Create Tag...` for a specific commit in the History tab.



Once the tags is pushed to GitHub/GitLab. You can create a release of the version by clicking `Release` on the `Code` page of your repository. There you can select a tag and generate release notes.

It’s recommended to add the relevant part of the changelog to the release notes on GitHub. An example can be found here

Publishing your book with a publisher

You can choose to publish your online book with a publisher. This could increase findability of your book, but may have consequenses on the ease of editing, choice of platforms, versioning restrictions, and copyright and licensing considerations.

For TU Delft employees, you can decide to publish via TU Delft OPEN interactive textbooks. Publishing will give you a copyright check, ISBN or DOI number, and registration in several shared databases but limits the amount of changes you can make to your book: for substantial changes a new published version is needed with an updated copyright checks. Small changes like typos can always be made, which are processed in the published version every hour. In case of editing a book for a course during that actual course and/or your book has limited value outside your course's content, we advise you only to publish an archived version of your book whenever the academic year is over.

Copyright and Licenses checklist

Contents

The previous sections have familiarized you with the necessary software for your user type, the basics of Git, and how to collaborate with your team using Git. Now it's time to create your own content. This section will provide an overview of the copy right laws which need to be taking into account when creating and including content at TU Delft!

Although the focus of TeachBooks is mostly to help the teaching staff create interactive textbooks for their students, it can be attractive to spread ones wings a bit and reach a wider audience. The checklist on this page provides a quick overview of the *minimium* information you should be aware of when getting started in terms of copyright, avoiding legal issues and making your life easier later on when/if you decide to publish your book with an open license, for example via TU Delft Open.

For more information specifically related to online books read the following pages; for more generic information about publishing visit the TU Delft Copyright website.

The Copyright Checklist

This checklist is meant to quickly illustrate the proper way of using the copyrighted work of others. In particular for online books that are available on the internet, it is important to be aware that publicly accessible content is already considered *published.*

Item 0: Your book is already published!

0) If your book is accessible online, it is already published!

This may surprise you, but *legally* speaking, once your book content is available online, it is technically already *published.* A quick search on the internet reveals most definitions involve two key words: to *issue* something for *distribution.* Thus, the act of sharing book material with students via the internet is a form of publishing. Of course this is different than doing so via an established publisher, for example: others may not be able to easily find the material, the product may be unfinished, mistakes may be present, etc. However, this means that *copyright ownership* and *authorship* has been established and must be considered (by you!) in at least two ways:

1. You are responsible for properly attributing the work of others included in your book, and
2. You should explicitly state how you would like others to (re)use the content in your book.

The remaining checklist items cover the first point, which is where the risk of legal and financial consequences for you and your employer lie. Continue with the following pages to find out more about the second point.

Items 1-4: Attribute the work of others properly

1) Your own new content

If you have created the content yourself, you are allowed to use it if there are no other agreements in place.

Note that this is specifically for *new* content; as in unpublished anywhere else! If you have already published it elsewhere, you should attribute yourself as if you would any other material.

2) Somebody else's content

If you want to reuse someone else's work (that is *not* released under an open license!) you need to ask for permission. When you ask for permission please do it in a written format and let the copyright holders know that you are going to use their work in an open licensed work.

If you are granted permission, be sure to exclude the work from the open licensed conditions specified in your book (typically easiest by stating so in the attribution/footnote/caption, as well in the colophon of the work). If you have asked for permission and received no answer, consider it as a negative response and abstain from using the work you were intended to.

If you really need to reuse someone else's work, please also check whether the copyright exception right to quote is applicable to you.

Here are a few selected answers from the Copyright Information Point:

- Citations of other work
- Use of copyrighted multimedia
- Use of copyrighted text.

3) Your own old content

Even if created by you, you need to attribute yourself in order to avoid self-plagiarism. If the material has an open license, this is straightforward. If there is no license, or a non-open license, you may need permission from your university (the copyright holder; explained here). Read more here

If you want to use your own content that a (commercial) publisher has published, you must check your contract to see if you are allowed to use it. Read more here.

Even if you have created the work yourself but years ago, check whether you have the right to reuse it. If you have created it while you were under the employment of your previous employer, then you will need to ask permission from them.

4) Open licensed work

Work by others that is provided with an open license can be used without asking for permission; if you find yourself considering whether to reuse someone else's work that does *not* have an open license, you should try to check if there is an alternative available that is provided with an open license (i.e., replace the work you wanted to reuse in the first place). It turns out there are many online platforms that can help you locate open licensed work.

If you use open licensed work, you need to attribute the license under which the work is licensed and you need to check the compatibility of the licenses. This means you need to check which license the work you want to reuse has as well as the license that your TeachBook will be provided with (more information is provided on the Licenses page).

When you want to adapt previously published work you need to ask permission for it. Otherwise please check whether you can find an open licensed work that you can adapt if their license it permits it.

A Final Note: Open Licenses Make Life Easy!

If it's not clear by now, hopefully you see the value of creating and using material with open licenses. This makes it much easier to reuse content, as the license itself allows for this, and you don't have to spend time asking for permission!

Find out more about open licenses specifically for books on the following pages.

Copyright Considerations

Contents

Copyright law dictates how online book material can and should be created, maintained and shared. An author should understand a few key concepts, as well as best practices, in order to create a book that not only is a useful resource, but, also fairly reuses the work of others and, if provided with an open license, enables original content to also be reused.

Before getting into specific license types and our recommendations for properly attributing and referencing the work of others, let's review a few key copyright-related concepts that are specific to online textbooks.

> **Tip**
>
> Here we only focus on aspects that are explicitly relevant to the creation and sharing of online books. Visit the TU Delft Copyright Information Point website for a more thorough explanation of these important concepts.

Copyright

The legal right of the content *owner* to use, publish, distribute, share, and/or duplicate the work. Copyright is divided into **legal rights** and **moral rights**, depending on the material and circumstances under which it was created.

As indicated below, it is the *moral rights* that give you, the author, the right to be named as the creator of the work.

Types of Material

It appears that copyright law distinguishes between two types of material (not including that of students):

1. Educational material
2. Articles, book (chapter), conference paper, etc.

We aren't really sure what the definition of *educational material* is, but since pretty much everything we do is for teaching students or teachers about things, we are pretty sure it includes all of it. Perhaps it is easier to see a few examples of what is *not* included: journal article, book (chapter), conference paper.

Warning

The TeachBooks Team isn't really sure when the "line is crossed" from educational material to book. More specifically, when the copyright owner is you (the author) or the University (your employer). For now, we assume that all books made by employees of a university are considered educational material.

Do you have a better answer than this? Let us know!

If you can read Dutch, the report *Leermaterialen kiezen* may shed some light on uncertainty about what is considered "educational material" (download PDF here).

You Don't Own Everything

The law is very clear on this, and the TU Delft Copyright page states it quite clearly:

> Copyright to educational material made by a TU Delft teacher, rests with TU Delft. Amongst the other moral rights, the teacher retains the right to be named.

In case that wasn't clear enough, *TU Delft owns the copyright,* not you! In other words, you should do this:

> by Last, First © TU Delft, Year

Don't do this:

> © Last, First, Year

We recognize that this may make you angry, and we can relate: as teachers we put a lot of effort into the material we create. However, we have come to see that this is *not* a horrible situation; in fact, it is a good one. As the creator of the material you still retain moral rights to be named as the author, and there are other benefits as well. For example, the ability of other teachers that may take your place after you are gone to legally use and build upon your work in the future.

Tip

One reason copyright lies with TU Delft is because educational material must remain available for students when the author leaves. Additional justification can be found in the Open Educational Resources (OER) Policy Document, here.

The use of open licenses actually does not prohibit the author from using the material in the future, even if not employed at TU Delft. For example, if you publish a TeachBook at TU OPEN, with a CC BY license, you are always entitled to "take" the book with you when leaving by simply changing the cover.

Still don't agree? Sorry. Please take this up with your nearest library or copyright expert and let us know if you get a different response! In the meantime, we will simply list ourselves as authors, use CC BY licenses and remember that TU Delft holds the copyright for our work produced as teachers.

> ⚠ **Warning**
>
> The TeachBooks Team isn't sure where code and data lie here. Since TeachBooks is oriented towards education, we assume that TU Delft also owns the copyright for this.
>
> Do you have a better answer? Let us know!

Licensing

This dictates how others may or may not be able to share or (re)use your work. It is (fortunately) becoming more common to include licenses with educational material (including books), which makes it much easier to understand how specific material can be used. Note that *in the absence of a license, you must assume that nothing can be reused!* In this case you have to ask permission to the copyright holder, which can be a time-consuming process. License considerations are described in more detail in the following pages.

> ⚠ **Warning**
>
> The TeachBooks Team isn't really able to explain why an author can dictate the license of material they create while TU Delft is the copyright holder. In other words, if I create a new assignment, I can choose to license it as CC BY.
>
> Do you know? Please get in touch with us and help!

Summary

There are many legal aspects to consider when creating an online book. Hopefully this page simplifies the story somewhat and indicates that there is a finite set of actions you should take to ensure that your book properly attributes and references the work of others, as well as indicates to others how they can use your work. This includes the following:

1. **Choose a license** and including with the *source code* and *website* of your book (we hope you choose CC BY!).
2. Clearly **indicate where and how you have used the work of others** in your book (i.e., within the *content* of your book).
3. **Summarize all sources** used in your book, as well as what may or may not be included under the license which you have chosen.

The following pages illustrate how to do this. It turns out to be not only a clear set of instructions for (re)users of your content, but also to help you remember what you have used in your book and where to find it later if changes are needed.

Licenses

Contents

Here we present an overview of various license types, with a particular focus on the preference of TeachBooks to use Permissive Open Licenses, regardless of whether material is *content, code or data* (described below).

> **Why permissive open licenses?**
>
> Since we (TeachBooks) authors have been working on making online books, we see the value in making content available for reuse. When preparing a new page or assignment, nothing makes us happier than seeing a CC BY license on something that we would like to (re)use ourselves. Why? Because we know that all we have to do is properly attribute the work and we can get on with our true passion: **teaching!**

This page is written expressly for content authors of online interactive textbooks (e.g., those including, but not restricted to, Jupyter Books), which are composed of two parts:

1. An online document in the form of a website, accessible at a specific URL, and
2. The *source code,* which defines the *content* and is required to create the website.

This generally does *not* include the software required to convert the *source code* in the the *website*.

> **Note**
>
> Because the source code of an online book defines the content, and only occasionally includes code required to create the website, one should consider the *source code* of a book as *content* and not *code*. When this is not the case, we recommend included a code-specific license and indicate the specific files to which the license applies in your repository.

If you are personally undecided on which license to choose for your work, we hope that the information in this chapter convinces you to also have a preference for permissive open licenses. If you are still unsure, we recommend reading the blog post by Hall (2021) that inspired this page. Another useful resource is the website choosealicense.com, which includes many more options than those described here. Finally, consider also the policies of your own university or organization; for example, TU Delft polcies can be found here.

License Types in Context

Have you ever been overwhelmed by the massive amount of information (and especially *opinions!*) available regarding licenses? We have too! However, you are in luck: by dividing all possible license types into *openness* and *material type,* it becomes much easier to understand the differences, and make a decision that fits your situation best.

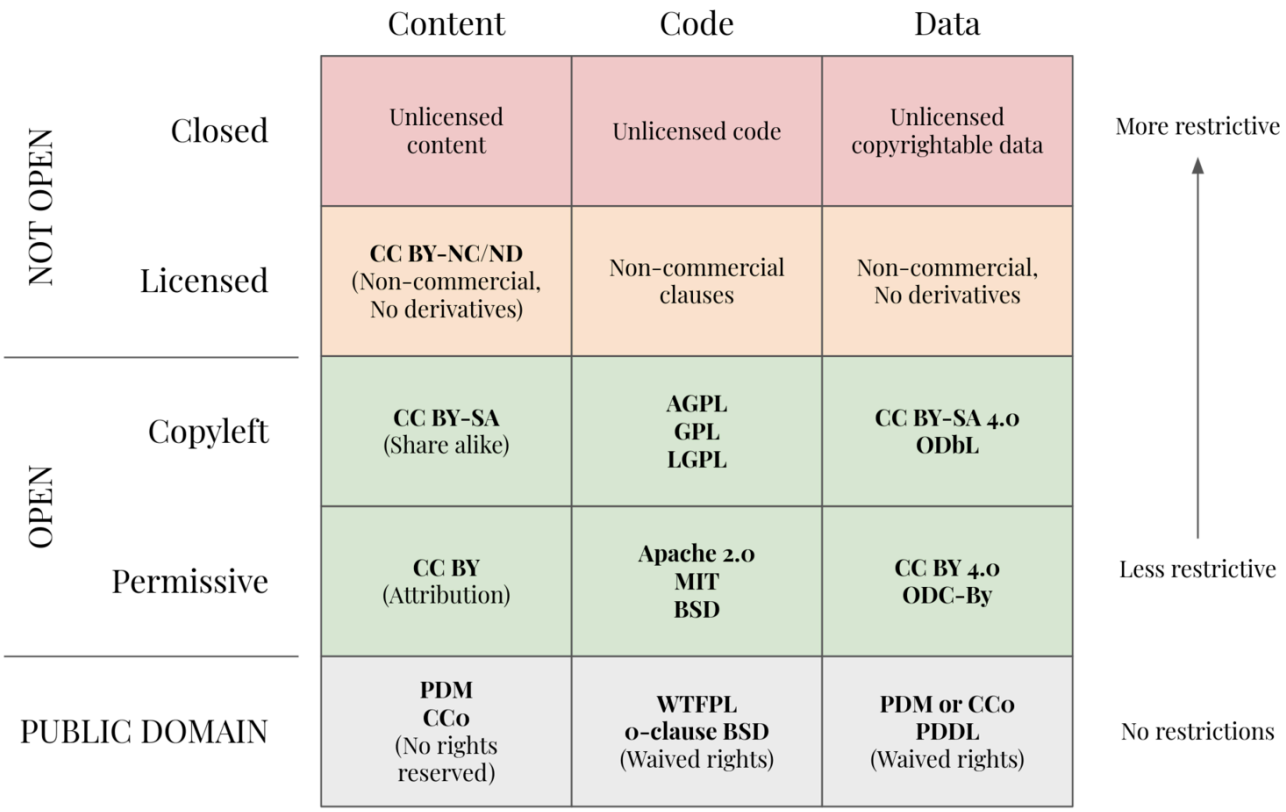


Fig. 86 Overview of license types, emphasizing "openness" and subdivided by material type: content, code and data. Source: Hall (2021), CC BY. ❝Find out more here.

Although this figure only includes a few different license types, the way of dividing by *openness* and *material type* can be applied to any license. This is especially useful when you are unsure which license to choose, or when you are considering a license that is not included in the figure. Remember to check the website choosealicense.com, which includes many more options than those described here. You can also an immediate comparison in the choosealicense.com appendix.

Open versus Not Open

We recommend using open licenses to allow other people to reuse your material, just like you'd like to reuse material from others.

Content, Code and Data

For content, we use a CC BY 4.0 license, for Code we primarily use the BSD-3-Clause license. Both of these are open to allow reuse.

A Note to the Wary

Are you thinking something like this?

> I've put a lot of work into my material, and I don't want people to take credit for it or use it without my knowledge. However, I do see the value of choosing a license, especially an open one. Maybe I will choose something like CC BY-NC-SA, which allows others to use my work but not for commercial purposes, and not allowing them to modify it. That way, I can still get credit for my work and others can use it for educational purposes.

This is a nice sentiment, but when it comes to books, we don't advise it for several reasons. The following sections anecdotally explain "not open" and "permissive" licenses are not preferred.

> **Note**
>
> Think you can improve on the examples described in the following sections? Make an Issue or Pull Request!

Why is ND not preferred?

The "ND" license prohibits derivative works: in other words, the material cannot be modified and must be used in its entirety. This is problematic for a few reasons:

- What if you have a wonderful page describing Topic X, but there are a few contextual phrases that are undesired, for example: "this is required reading for Tuesday, December 17, 2024 in CIEM1000." That would be distracting for your own students, but you can't remove it.
- What is the point of including a website together with another website? It's easier to just link to your material directly.
- What if someone only wants a small part of your book? They can't use it.
- What if someone else makes similar material and releases it under CC BY and you include it in your own book under an ND license? It seems unfair not to also provide them an opportunity to benefit from your work. In addition, this is not allowed! You can't put work under a more restrictive license (see also compatibility table below).
- Often someone creates a copy of your work and makes improvements. The Git system (e.g., forks, commits, pull requests, etc) make it very straightforward to also include these improvements back into your original work. This would not be possible if you did not allow for reuse in the first place!

Why is NC not preferred?

The "NC" license prohibits commercial use. It sounds nice, right?

> Great, I can guarantee that big for-profit companies can't use my work for their own gain. This is important to me at a public or non-profit institution.

To be honest, it is more trouble than it's worth, primarily because the law seems to be vague on what constitutes commercial use. Many universities require tuition to cover operating expenses, which is often broken down into by number of students, credits, courses, etc. This could be interpreted as a form of commercial use and thus prevent other educators from using your work, which is precisely the opposite of what you were trying to facilitate!

Why is SA not preferred?

The "SA" license requires that any derivative works be released under the same license. If you have two sources, each of which has a different SA license, how can you include your work that incorporates that content under the same license? You can't! Once again, if your goal is to convince others to share your work in a responsible way, your effort is thwarted. As with NC, this type of license seems to be more trouble than it's worth. A similar story holds for the other 'copyleft' lincenses, as all copyleft licences require that the original license is perserved in derivative work to some degree.

Summary of License Type Compatibility

The previous sections can be illustrated by examining a so-called "license compatibility chart," which illustrates whether or not you can reuse material from a specific license type in another. Upon examination, you will see that CC BY is the most

permissive license, both in terms of what you can do with the work of others, as well as what others can do with your work. We hope you will consider this when choosing a license for your own work!

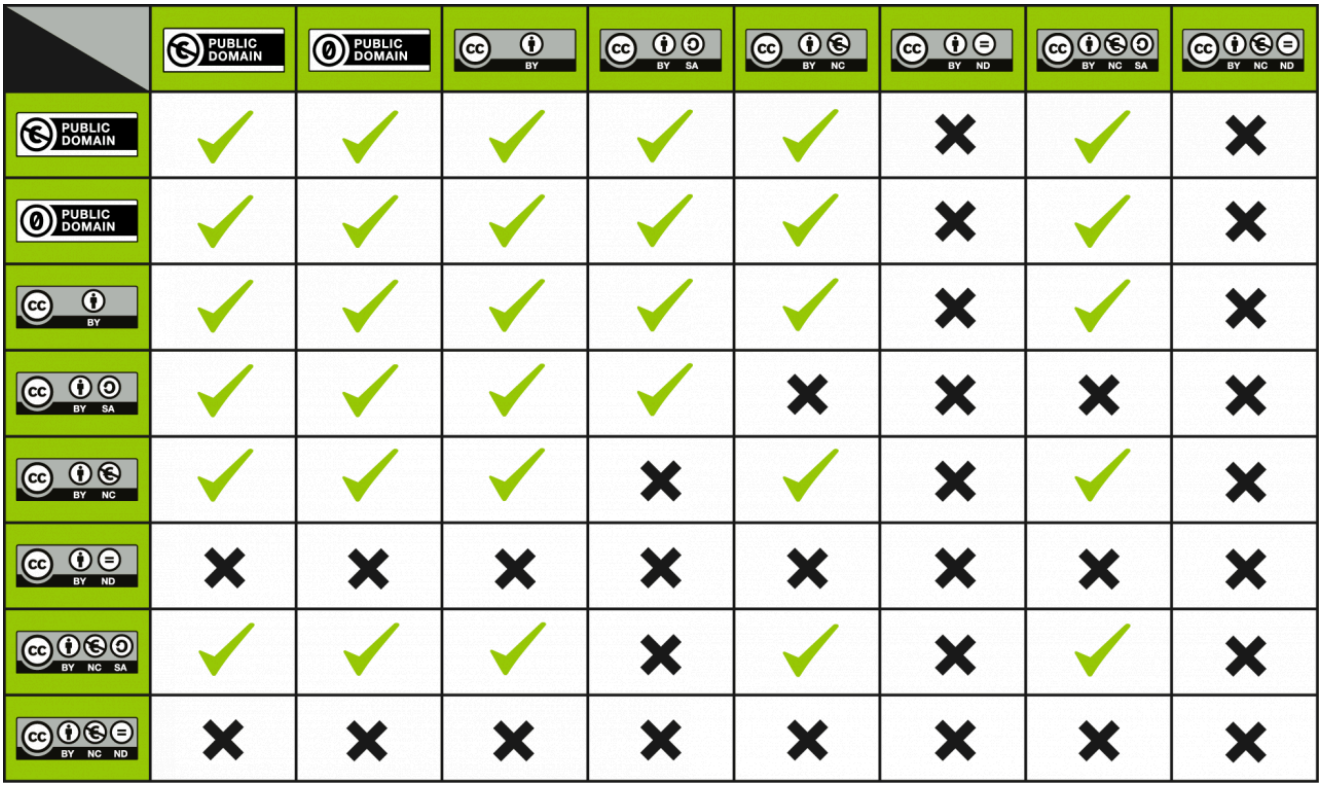


Fig. 87 Compatibility between Creative Commons license types. Source: Kennisland, CC0. ❝Find out more here.

Examples from Related Applications

> **Warning**
>
> Work in progress.

Code in a Book Repository

For example, a simple script to parse files in your repository. You can include a special license with this code.

Educational Materials with Code

What if your *content* includes code. This happens in many science and engineering courses where programming is a key part of the learning process.

For example, the MUDE Team is working towards releasing the files associated with assignments under an open license. Since the majority of the content is written in a storytelling style in Jupyter notebooks, a CC BY license is probably the best choice if only one license is applied to a single file. However, if released in bulk it may be better to follow the advice of Matt Hall:

> You have to be practical; maybe it depends on whether you consider your notebooks to be 'content' or 'source code'. I sometimes put at the bottom of a notebook something like **Open source content. Text is CC-BY, code is Apache 2.0** and I think this makes my intent clear.

Recommendations

Contents

As authors, one important set of objectives is to make sure that we:

1. Avoid plagiarism and copyright infringement,
2. Properly attribute (reused) content, and
3. Indicate to readers where to find the original content,
4. Create a record for ourselves that clearly documents what we have used.

As item number 3 aligns with the standard approach of citation and references in academic writing, much of the work is already quite clear. At TeachBooks we use APA guidelines by default; instructions for implementing this in your book can be found on this page.

Addressing Items 1 and 2 can be overwhelming at first, but we have tried to illustrate them clearly in this manual. In particular, the sections in this chapter, especially this page, which collects best practices used by TeachBooks collaborators, along with explanations for why we do it that way. We try to present a standard way of referencing material and attributing authors clearly; however, this can vary widely based on your own content, book setup and discipline.

The recommendations illustrate:

1. An efficient and consistent approach to accomplish the objectives above, and
2. How you can implement this easily in the *source code of your own book*.

We believe the approach outlined here is also a useful way for you as an author to *remember which material you have used in your book and find or update it later if changes are needed.* This is Item 4 in the list above, and hopefully avoids frustration in the future if you need to revise your book after a long period of inactivity and cannot remember what content was written by you, or written by others.

The recommendations on this page are designed for content that is available in two formats: *source code* and a *public website*. Additionally, it is focused primarily on file formats and open content relevant to Jupyer Books, which means content written primarily in *text files* (e.g., .md, .ipynb, etc.), also called *source code,* shared with an open license and used to create a *website* (e.g., HTML files accessible at a specific URL). Software is used to convert the source code into a website, however, this is generally not included in the creative content shared using the book license.

Key Locations in the Book

Note that there are several types of pages and files that can be incorporated into your TeachBook which, together, accomplish our goals of proper attribution and referencing. These are summarized in the table and explained in more detail in the following sections, along with examples for implementation in a Jupyter Book.

Table 1 Key Locations in the Book for Attribution and Referencing Best Practices

Page Name	File Name	Example in this book	Purpose
Content Page	`*.md`, `*.ipynb`, etc.	Licenses page	Identifies source and links to credits page.
Credits and License Page	`credits.md`	Credits page	Collets license and copyright information in one place.
References Page	`references.md`	References page	Documents all sources, as in any other published work.
License Page	`LICENSE`	File stored in repository here.	Document license terms; includes licenses of reused content.

Content Pages

Make sure you show the source of your content where it is provided. We adopt a standard icon and link to help readers recognize this: ❝Find out more here.. This can be used to provide attribution to authors for specific pages too

How to implement this in your book

Some examples can be found in this manual:

- Releasing book online: a content page which is taken fully from another repository
- Figure Overview of license types, emphasizing "openness" and subdivided by material type: content, code and data. Source: hall21, CC BY. Find out more here.: a single licensed figure added to a page taken from another website
- Copyright and Licenses checklist: a content page which adapts material from another website

Format as citation

Copy the following line into your content file

```
> Written by `<author(s)>`
> This page reuses <license> content from {cite:t}`bib_id`. {fa}`quote-left`{ref}`Find out more here.<link to external r
```

This produces

> This page reuses CC BY content from Moore (2023). ❝Find out more here.

Format as custom admonition

The previous syntax might be used for other purposes as well. An alternative is a custom attribution admonition:

Therefore, use the following syntax:

````
````{margin}
```{attributiongrey} Attribution
:class: attribution
Written by <author(s)>
This page reuses <license> content from {cite:t}`bib_id`. {fa}`quote-left`{ref}`Find out more here.<link to external res
```
````
````

To make this possible we use the custom admonitions of the custom named colors sphinx extension in combination with a custom css file in `book/_static/` and the line `named_colors_custom_colors: {'attributiongrey':[150,150,150]}` in `book/_config.yml` under `sphinx: config:`

Credits and License Page

This place describes all relevant information in one place regarding copyright and licenses, including reused content. This page should also be useful to the author returning to the book after a long period of inactivity.

How to implement this in your book

That's easy! Just copy the page from out template and adapt to your purposes.

References Page

Lists all references used in the book, including those not directly cited in the text (e.g., reused material).

A special case is made for material licensed under the public domain, for example, the "no rights reserved" CC0 license. As these can include a large number of single files (e.g., images), each with their own link and author, by default we do not include these in the References Page, but we do list the source in the credits. This is especially useful for finding the material later, or allowing anyone who would like to reuse your content to find and cite the public domain source material directly.

One example in this manual is Compatibility between Creative Commons license types. Source: Kennisland, CC0. Find out more here., listed on the credits page here.

How to implement this in your book

Proper referencing typically involves setting up an entry in your `*.bib` file and using a citation key in the text on the content and Credits pages, which can be handled automatically if you are using the APA references tool. This also allows you to use the text (`t`) and parentheses (`p`) citation styles.

```
{cite:t}`cite_key`
{cite:p}`cite_key`
```

which produce these citations: text by Moore (2023). And parentheses (Moore, 2023) when using APA references tool.

License File

Specifies license and copyright of book, along with licenses of reused content also included in source code.

Entire file only included in source code. License type stated on Credits page and included in footer.

How to implement this in your book

Create a LICENSE file in the root of your repository (note that there is conventionally no extension in the file name). Platforms like GitHub and GitLab can help you with this via the home page of your repository (look for a button that says "Add a License").

The file should contain the full text of the license you have chosen for your book, along with any other licenses that apply to content you have reused. For example, the TeachBooks Manual uses a CC BY 4.0 license, which is included in the LICENSE file in the source code at github.com/TeachBooks/manual.

Note that although most of the license text can be created from a template by GitHub, you may still want to identify yourself as the author and/or the copyright holder at the top of the file. This provides clarity to readers and, if reusing your work, it is also easier for others to identify your license when including it in their own source code.

Special Cases

Is there a special case you would like to know more about that is not covered here? Please let us know by creating an issue in the appropriate TeachBooks GitHub repository.

It can also be very useful to check what others commonly do in the open source community. Note, however, some discretion should be applied, as not everything you see on the internet is done properly. Our advice is to find a few good examples of people you can trust and copy them.

For example, when Robert and Tom are in this position, they often wonder WWJD? (What Would Jason Do?) Jason Moore has been making and (re)using open source material for a long time, and his GitHub repositories and websites at github.com/moorepants are a great source of inspiration!

Overview

Contents

As TeachBooks, we collect a suite of existing open-source software so you don't have to! Some of the software is developed with our TA's to improve the learning experience of our students and ease the book-development process for our teachers. As the open-source software landscape changes rapidly, it is essential to keep in contact and share resources amongst ourselves to minimize maintenance and downtime for our book websites and focus on what really matters: teaching!

Since the list of TeachBook features is getting quite long, we have grouped them in categories:

- Original Jupyter Book and Sphinx features
- Easy editing process
- Additional functionality
- Book styling
- TeachBooks student-view features

Additionally, not all features are built and shared in the same way. We do our best to make sure that as many tools as possible are included automatically when using our TeachBooks Template Book; if you are not using the Template, we try to make sure each of our tools can be used independently. For transparency, tags will help differentiate between the different backgrounds of the features:

- **Javascript overlay**
- **Chrome Extension**
- **GitHub App and Javascript script**
- **Python Package: teachbooks**
- **GitHub Reusable Action**
- **GitHub Template**
- **Git Workflow**
- **Sphinx Extension**
- **iframe**

Finally, the purpose, installation process and usage of each features is elaborated on in the respective sub-sections.

To see examples for these features, go to the Examples chapter.

As the TeachBooks Python Package and the Deploy Book Workflow are important tools that incorporate and deploy more than one feature, an additional explanation is provided on this page in more detail, with links to the pages in this manual where individual features are described.

TeachBooks Python Package

The TeachBooks Python package is a collection of tools that are used to enhance the Jupyter Book software package by adding features and making customization easier. In general, it is only of interest to user types 4 and 5 when building a

book locally. However, it may be important for other users to know what the package does, as it is incorporated in the Deploy Book Workflow and, therefore, any book created from the TeachBooks Template. This raises two important points:

1. If you are a book user using the TeachBooks Template and/or the Deploy Book Workflow, your book is by default built with the `teachbooks` package, and the way your book is built and features that are included in the book may change automatically when it is updated (these will be non-breaking changes; if not, we will notify our mailing list).
2. If you are using the `jupyter-book` package to build your book, some of the features described in this Manual may not be available to you (i.e., those listed on this page). This will be the case if you are using a setup described in the Jupyter Book Manual, for example, this GitHub workflow.

For those who wish to use the package, it is available on PyPI and can be installed using `pip install teachbooks`. As `teachbooks` is a wrapper, the package is meant to replace the usage of `jupyter-book` commands (although it does indeed add a few new commands to the CLI, e.g., `serve`). It is conventionally used in an identical way, as follows:

```
teachbooks build book
```

This Manual describes the main features and usage of the package. For those who wish to understand more of hte technical details, note that the implementation the API is is documented at teachbooks.readthedocs.io/.

List of Features

Items in the list here are currently implemented in the TeachBooks package and described on other pages of this manual:

- Draft-Release Workflow
- External Contents
- Local Server

The package is also set up to handle APA references when using the Draft-Release Workflow (the `_ext` subdirectory and its contents are manually copied during the pre-build phase). This will not be necessary once the APA Reference feature is turned into a proper Sphinx extension or Pybtext plugin.

Deploy Book Workflow

The Deploy Book Workflow is by default incorporated into any book that has been created using the TeachBooks Template. We also strongly encourage anyone to consider this tool as an alternative to the "standard" workflow provided by Jupyter Book, as ours automatically builds a different version of your book for every branch in your respository, along with a number of settings that can be used to customize the build process and URL structure of your book(s). This tool is described in more detail on the Deploy Book Workflow page of this manual, and one should note that it influences a number of other aspects of a TeachBook.

Related Features

A list of tools and features that are dependent on, or related to, the Deploy Book Workflow (DBW) is provided here:

Important for these features:

- Releasing book online: the main DBW page
- TeachBooks template: books created from the Template include DBW by default
- Draft-Release Workflow: can be configured directly using the DBW
- Auto-updating packages: behaviors of this tool may influence the DBW

- TeachBooks Python Package: included by default in the DBW
- Build pull requests from forks: created to provide DBW-like behavior for PR's from forked repositories
- APA References: a temporary fix is included in `teachbooks` to enable use in the DBW

Book Build Commands

By default the DBW builds books using the following command:

```
teachbooks build book/
```

To use the Draft-Release Workflow with the Deploy Book Workflow the `BRANCHES_TO_PREPROCESS` variable must be configured for a specific branch, which will then execute the following command:

```
teachbooks build --release book/
```

Environmnets and Caching

As described on the DBW page, there is a lot going on "under the hood" with regards to caching of GitHub Action artifacts. This can lead to undesired behavior when using the DBW, especially if package version numbers are not precisely defined. In general we expect the risk of this occurring to be low, as it should only happen when multiple branches are being actively used (new commits on each branch at least once per week), a package from the `requirements.txt` file is updated in the time between creation of two or more branches, and that package *also* has a significant impact on the book building process. As the Python virtual environment cache is replaced by default if older than one week, the issue should resolve itself within that time frame.

Note that Specifying package version numbers explicitly and updating them via Dependabot is an excellent way to ensure that environments in the DBW are always up to date and this issue is avoided.

If you are *not* using Dependabot and are not able to get your packages updated when the DBW is running, delete the cache manually and rerun your GitHub Actions job (go to Actions tab, then looking for the "Caches" section under the "Management" pane on the left hand side of the screen).

Original Jupyter Book and Sphinx Features

These pages will provide a list of some common, basic book features which are all available in the original jupyter{book} package. Additionally, this page includes also best-practices.

Anatomy of a Jupyter Book

Contents

To build a Jupyter Book, you need three things:

1. A configuration file
2. A table of contents
3. Content

The configuration file

The configuration file of Jupyter Books is named `_config.yml`. It mainly contains settings that apply when building the book. Here is a basic example of a configuration file, taken from this book:

```yaml
title: Jupyter Book Manual
author: Interactive Textbooks CiTG
logo: images/TUDelft_logo_rgb.png
copyright: CC BY-NC

execute:
  execute_notebooks: auto

sphinx:
  extra_extensions:
  - sphinx_inline_tabs
```

Technically speaking, a `_config.yml` file is not required to build a Jupyter Book. If you don't make one, `jupyter book` will just use all default values. However, you should make a configuration file which includes at least the following settings:

- `title`: the title of your book, which appears on the top-left of every page, under the logo.
- `author`: authors of the book, which appears in the bottom margin of every page.

- `logo`: (relative) path to the logo of your book (optional).
- `copyright`: the licenses attached to your book.

In addition, the following option can also be useful:

- `execute_notebooks`: *turn on/off* the execution of Jupyter Notebooks during the build process. *On* by default. If you perform heavy computations in your notebook (machine learning, FEM models, et cetera), you might be better off running the notebooks on a more powerful machine as opposed to the CI/CD server. To turn it off, specify the value `'off'`. You can also exclude specific notebooks by creating exclude patterns in the filenames. For more info, see the Jupyter Book documentation.

> ⚠ **Warning**
>
> The configuration and table of contents files are in YAML format (short for YAML Ain't Markup Language). YAML has a specific syntax, which can cause some errors if you don't adhere to it. Just like Python, indentation is very important. You can find an overview of the syntax here.

The table of contents

The table of contents (`_toc.yml`) file is where you define the structure of your book. You can organize content in *parts*, *chapters*, and *sections* (and subsections, each new section creating a dropdown menu in the ToC). This is what a table of contents could look like:

```
- format: jb-book
- root: intro

chapters:
- file: chapter_1
  sections:
  - file: section_1_1
  - file: section_1_2
- file: chapter_2
  sections:
  - file: section_2_1

et cetera...
```

`root` is the landing page of your book. It will be the first page that people see when they visit the book. You can include files by providing their *relative* path (so the location with respect to `_toc.yml`). These files can be a combination of Markdown, Jupyter Notebooks,

and Restructured Text. The example above does not have parts, so the menu on the left will consist of just a list of the chapters. You can also group chapters in parts like this:

```
- format: jb-book
- root: <homepage>

parts:
- caption: Part 1
  chapters:
  - file: chapter_1
    sections:
    - file: section_1_1
    - file: section_1_2
      sections:
      - file: subsection_1_2_1
      - file: subsection_1_2_2
  - file: chapter_2
- caption: Part 2
  chapters:
  - file: chapter_3
    sections:
    - file: section_3_1

et cetera...
```

Content

The main thing people are interested in, is the content of the book. To help the reader, you can structure the book into chapters, each which sections and subsections as explained below.

The structure of a chapter

The 'nested' structure in the TOC is one way to organize your book. Another way to do so is in the file itself. The structure is defined by the number of `#`:

```
# Chapter 1 title

## Section 1.1

### Subsection 1.1.1

## Section 1.2

### Subsection 1.2.1
```

The depth for numbered subsection can be set in the TOC file:

```
parts:
- caption: Part 1
  numbered: 2
  chapters:
  - file: chapter_1
    sections:
    - file: section_1_1
    - file: section_1_2
```

Including a chapter twice

In some cases you want to include the same chapter in two different places in your book (for instance parts). However, you do not want to make the same adjustments in different files, the content should only 'live' in one place. The solution is to use `include`:

````
```{include} Argument
```
````

The *argument* in this case is the location of the file you want to include (for instance: `/basic-features/equations.md`). Note that when building the book locally, you get some warnings for duplicate labels..

We provide an example of the use of `include` below: We included the chapter equations in this page.
Equations can be included in two ways: inline or display mode.

To make an inline equation, just put the LaTeX equation between $-signs. For example, `$F = m \cdot a$` produces: $F = m \cdot a$.

Display mode equations (the ones that take up a whole line), can be inserted using double $-signs, like this:

```
$$
    F = m \cdot a
$$
```

which will produce:

$$F = m \cdot a$$

Tip

The scientific rules dictate that quantities (F, m, a etc.) should be printed in italics. Units, however, should not! One can use `\text{m}` in an equation to comply to this rule:

$$F = m \cdot a = 10 \cdot 9.81 = 98\text{N}$$

Warning

Make sure there is a blank line *before* the display mode equations, otherwise it will render as inline and display the outer set of $$ symbols. Also, the Euro symbol is not included in MathJax (see note below) and must be specified using `\unicode{0x20AC}` (note that it displays incorrectly in some Markdown renderers like VS Code).

To number the equations and refer in text, you need to provide a label to the equation. Just put the label between brackets and place it after the last $$, like this:

```
$$
    F = m \cdot a
$$ (newtons_second_law)
```

Resulting in:

$$F = m \cdot a \tag{1}$$

Equation (1) can now be referred to.

Note

The Jupyter Book uses MathJax to display equations, which provides some LaTeX-like functionality, but not all! For example, the Euro symbol is missing, and packages like `siunitx` are not available (hint, use `\textrm{}` and `\textrm{}`). When something is not working, it's useful to search for MathJax-specific solutions (hint, include "mathjax" in your Google search).

Figures

To add a figure, just copy the figure to the correct directory. Then, in your markdown file, include the figure as follows:

```
```{figure} <figurename>.png/.jpg
---
height/width: <height or width in pixels>
name: <name of the figure>
align: left / center / right
figclass: left blank or "margin"
---
<Figure caption>
```
```

For example:

```
```{figure} ../images/TUDelft_logo_rgb.png
---
width: 70%
name: demoexample1
align: center
---
example 1: width (70%) as percentage, align center
```
```

provides this output, which looks nice.

Fig. 88 example 1: width (70%) as percentage, align center

Using the name label, you can refer to the figure as done with Figure 88.

You can find more documentation on including figures here.

There are many settings, for instance aligning right:

````
```{figure} ../images/TUDelft_logo_rgb.png
---
width: 50%
name: demoexample2
align: right
---
example 2: width (50%) as percentage, align right
```
````

with the result:

Aligning right can come with the potential downside (not always wanted) that text is wrapped around the figure. To avoid this one can use `<div>` elements, both before and after the text (make sure to leave an empty line after the `<div>`!):

Fig. 89 example 2: width (50%) as percentage, align right

````
<div style="clear: both;">
and the output
</div>
<div style="clear: both;">

```{figure} ../images/TUDelft_logo_rgb.
---
width: 50%
name: demoexample3
align: right
---
example 3: width (50%) as percentage, a
```

</div>
````

with the result:

Fig. 90 example 3: width (50%) as percentage, align right with additional div style

including `figclass: margin` sets the figure to the column at the right.

````
```{figure} demo97/demo97_figure1.jpg
---
figclass: margin
width: 50%
name: demoexample2

---
example 2:including figclass: margin
```
````

Equations

Equations can be included in two ways: inline or display mode.

To make an inline equation, just put the LaTeX equation between $-signs. For example, `$F = m \cdot a$` produces: $F = m \cdot a$.

Display mode equations (the ones that take up a whole line), can be inserted using double $-signs, like this:

```
$$
    F = m \cdot a
$$
```

which will produce:

$$F = m \cdot a$$

> **Tip**
>
> The scientific rules dictate that quantities (F, m, a etc.) should be printed in italics. Units, however, should not! One can use `\text{m}` in an equation to comply to this rule:
>
> $$F = m \cdot a = 10 \cdot 9.81 = 98\text{N}$$

> **Warning**
>
> Make sure there is a blank line *before* the display mode equations, otherwise it will render as inline and display the outer set of $$$ symbols. Also, the Euro symbol is not included in MathJax (see note below) and must be specified using `\unicode{0x20AC}` (note that it displays incorrectly in some Markdown renderers like VS Code).

To number the equations and refer in text, you need to provide a label to the equation. Just put the label between brackets and place it after the last $$, like this:

```
$$
    F = m \cdot a
$$ (newtons_second_law)
```

Resulting in:

$$F = m \cdot a \tag{2}$$

Equation (1) can now be referred to.

> **Note**
>
> The Jupyter Book uses MathJax to display equations, which provides some LaTeX-like functionality, but not all! For example, the Euro symbol is missing, and packages like `siunitx` are not available (hint, use `\textrm{}` and `\textrm{}`). When something is not working, it's useful to search for MathJax-specific solutions (hint, include "mathjax" in your Google search).

Videos

Videos uploaded to YouTube can be embedded in the Jupyter Book. There are several ways to do so:

1. To embed them in the Jupyter Book, first obtain the embedding link of the video. In order to do so, go to the *YouTube* page of the video (so not the Brightspace page), then click *share* in the description box. There should be a button *embed*, click that. Copy the HTML code that appears in the panel. Then, to embed the video, use the following

```
    <iframe
    width="560"
    height="315"
    src="https://www.youtube.com/embed/UCb-b82tzLo?"
    align="center"
    frameborder="0"
    allowfullscreen
    ></iframe>
```

The src can be used in combination with Iframes:

````
```{video} https://www.youtube.com/embed/UCb-b82tzLo?
```
````

Or the HTML-iframe code can be directly included in the markdown file.

Resulting in the video below:

2. Another option is to use a python coding cell. As this code cell should be run when the book is made, you have to change the config file and set `execute_notebooks:` to force. This comes with the downside that it takes considerable more time to deploy the book.

````
```{code-cell} ipython3
:tags: [remove-input]
from IPython.display import YouTubeVideo
VideoWidth=600
YouTubeVideo("YDBr1Lof_mI", width=VideoWidth, align='center')
```
````

Moreover, it requires one to have this code at the top of your markdown file:

```
---
jupytext:
text_representation:
    extension: .md
    format_name: myst
    format_version: 0.13
    jupytext_version: 1.10.3
kernelspec:
display_name: Python 3 (ipykernel)
language: python
name: python3
---
```

Note that this is not needed when you use a .ipynb (jupyter notebook) file.

References

To make references to figures, equations et cetera, use the following syntax:

- For figures, use: `{numref}`Figure {number} <name of the figure>``
- For equations, use: `{eq}`<equation label>``
- For citations, use: `{cite:p}`<bibtex_entry>`` for a citation between parenthesis, or `{cite:t}`<bibtex_entry>`` for an inline citation.

Examples

- Using `{numref}`Figure {number} <fig-gitpush>`` produces the output: Figure 92.

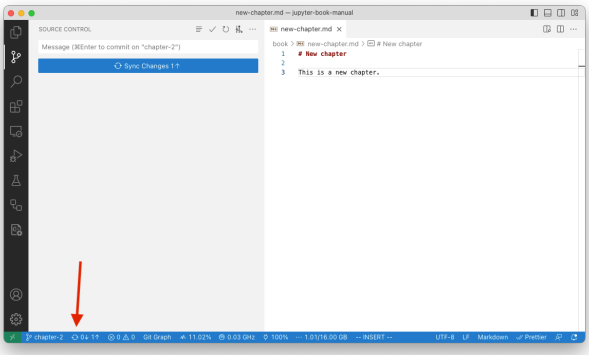


Fig. 92 Some caption here

- Using `{eq}`<eq:Newton>`` to refer to Newton's second law (3).

$$F = m \cdot a \tag{3}$$

Code

There are two main ways to include code: directly within a Markdown file or within a Jupyter notebook. More methods are available, but we don't include them here. The first option is great for including simple calculations, or generating simple figures when an image file is not practical. *More will be added later.*

Note that if you are using a `*.ipynb` file or including a code snippet in a `*.md` file, including a blank line between the text and the closing three tick marks will generate an empty code box of one line in the Jupyter Book.

When making notebooks for the Book, you might want to hide certain cells from the reader. For example, when including a simple figure generated from code or making a JupyterQuiz, we have to execute a code cell that generates the quiz. This code cell is ugly and distracting, so we do not want to render this in the final book.

We can change how the compiler treats notebook cells by using cell tags. You can find a detailed explanation on cell tags here. Specifically, have a look at the sections on *hiding* cell inputs and *removing* cell inputs.

The workflow of editing cell tags depends on your editor. If you're using Jupyter Lab, you can find instructions here.

For example, the following tag is used to convert the code input into either a drop-down (`hide-input`) or make it invisible (replace `hide-input` with `remove-input`). Replace `input` with `output` to do the same with the cell output:

```
{
    "tags": [
        "remove-input"
    ]
}
```

Once a correctly typed tag is added to the notebook it will appear in the buttons at the top and can be added to other cells. Multiple tags can be added to the same cell, in which case the tags are separated by commas.

Code blocks that produce figures

This can be done with notebooks, but it's not easy to make references to figures in notebooks. You can also place the code that produces and saves the figure in a separate `.py` file, and include that in a markdown file. Let's say we want to make a simple sine wave:

1. Place the code of the figure in a `.py` file. In this case, `sinewave.py` produces our figure. The code looks like this:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, endpoint=True)
y = np.sin(x)

plt.figure()
plt.plot(x, y)
plt.title('$y=\sin(x)$')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.savefig('sinewave.svg')
```

> **Warning**
>
> It is important that the name of the saved figure is **exactly** the same as the name of the Python script that generates it. Otherwise, the figure will not be generated by the Runner.

2. Test your code and make sure that the figure looks good. When you are ready to commit, place your code file in `book/code`.

The book uses a Matplotlib style sheet; to see what this looks like on your local machin you will have to run `plt.style.use('../../config/matplotlibrc')` after importing Matplotlib. This should be done outside of the `*.py` file you are creating. Note that the example style sheet path assumes you are working in the `book/code` directory. It is recommended to preserve a list of figures using a notebook in `code_checks`, where there are also working examples. It is not necessary to commit `*.svg` files as these are cleaned when the book is generated.

3. Now include the code file by using the following directives:

````
````{toggle}
```{eval-rst}
.. literalinclude:: ../sinewave.py
   :language: python
```
````
````

> **Note**
>
> the outer most directive needs an extra tick mark for nested cases.

4. Include the figure as described above. Remember that figures are saved in the `book/figures` directory. We also use a naming convention of `*_py.svg` for figures generated from code, to easily include them in `.gitignore`, since they are built in the CI/CD pipeline.

Badges, Buttons & Icons

Contents

jupyter{book} provides a range of features which are visually appealing and functional. These elements can enhance the interactivity and visual design of your book, offering additional information or links in a compact and visually engaging way. The badges, buttons and icons are made available through Sphinx and they themselves wrote a documentation about it.

Badges

Badges are small visual indicators often used to convey concise or important information. They can be very useful for categorising or tagging content for the reader like has been done for the subchapters under the features chapter of this manual.

There are three types of badges: plain, link and reference (for cross references within book) which are fully customizable and can simply be added with Markdown syntax:

Type	Badge	Code Syntax
Plain	plain_text	`{bdg-primary}`plain_text``
Link	https://teachbooks.io/	`{bdg-link-primary}`https://teachbooks.io/``
Hidden Link	TeachBooks	`{bdg-link-primary}`TeachBooks <https://https://teachbooks.io/> ``
Reference	Badges	`{bdg-ref-primary}`my_ref``

Here's an overview of all the available colours:

Full Colour	Code Syntax	Colour Outline	Code Syntax
my_text	{bdg}`my_text`		
my_text	{bdg-primary}`my_text`	my_text	{bdg-primary-line}`my_text`
my_text	{bdg-secondary}`my_text`	my_text	{bdg-secondary-line}`my_text`
my_text	{bdg-success}`my_text`	my_text	{bdg-success-line}`my_text`
my_text	{bdg-info}`my_text`	my_text	{bdg-info-line}`my_text`
my_text	{bdg-warning}`my_text`	my_text	{bdg-warning-line}`my_text`
my_text	{bdg-danger}`my_text`	my_text	{bdg-danger-line}`my_text`
my_text	{bdg-light}`my_text`	my_text	{bdg-light-line}`my_text`
my_text	{bdg-muted}`my_text`	my_text	{bdg-muted-line}`my_text`
my_text	{bdg-dark}`my_text`	my_text	{bdg-dark-line}`my_text`
my_text	{bdg-white}`my_text`	my_text	{bdg-white-line}`my_text`
my_text	{bdg-black}`my_text`	my_text	{bdg-black-line}`my_text`

Buttons

Buttons provide a way to create clickable elements that are more attractive than links, for example, to link to key sections of your book such as downloads or external resources. Jupyter-Book supports buttons using Markdown making them highly customizable. It is possible to link to external websites as well as chapters within your book. To link to an external page, use `{button-link}`. `{button-ref}` must be used when linking to a part within your book.

The basic code syntax for creating a button is as follows:

https://teachbooks.io

````
```{button-link} https://teachbooks.io
:color: primary
```
````

Overview

Overview

````
```{button-ref} feature_overview
:color: secondary
```
````

Buttons can be customized in markdown in a similar way as figures. You can experiment with these styles to create buttons that align with your book's theme. Here are the set of parameters which can be customized:

Parameter	Options	Functionality
color	primary, secondary, success, danger, warning, info, light, dark, muted	Set the color of the button (background and font)
outline	/	white button, coloured outline
align	left, right, center	Align the button on the page
expand	/	Expand to fit parent width
click-parent	/	Make parent container also clickable
tooltip	/	Add tooltip on hover
shadow	/	Add shadow CSS
class	/	Additional CSS classes

Finally here are some examples:

TeachBooks

````
```{button-link} https://teachbooks.io
:color: warning
:shadow:

TeachBooks
```
````

TeachBooks

````
```{button-link} https://teachbooks.io
:color: success
:expand:

TeachBooks
```
````

Icons

Icons can be included to visually represent actions or categories. They can take many shapes, such as an image or button inserted in-line within a text paragraph. Icons can be integrated using HTML syntax.

To add an icon, use the following syntax:

1. Image Icons

Let's have a look at the first sentence of this chapter again: jupyter{book} provides a range of features which are visually appealing and functional.

Here the Jupyter-book logo was inserted for a more professional look. It's including a link `href` and and image saved as `../images/logo-wide.svg`

```
<a href="https://jupyterbook.org/"><img  style="display:inline-block; height:1.5em; width
```

2. SVG Icons

A library of SVG icons can be found in the Bootstrap library.

By scrolling through the available icons in the library you can find just about any icon. Once selected, you need to copy the SVG of the icon to include it in the `path` in the HTML code. This paragraph includes a globe: 🌐

```
<svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class=
<path d="M0 8a8 8 0 1 1 16 0A8 8 0 0 0 1 0 8m7.5-6.923c-.67.204-1.335.82-1.887 1.855A8 8 0 0
</svg>
```

If you want to keep your files clean, you won't like these svgs. However, your icons is probably available in one of the following icon providers.

3. GitHub Octicon Icons

GitHub icons like can be added using the syntax: `{octicon}`mark-github`

Where `mark-github` can be replaced by any of the name of the icon provided here

By default the icon will be of height 1em (i.e. the height of the font).

4. Material Design Icons

The use of Material Design Icons like is explained here

5. FontAwesome Icons

The use of FontAwesome Icons like is explained here. There's no need to add the FontAwesome CSS.

6. Combining Buttons and Icons

Icons can also be included within buttons using the icon font for a modern, professional look. The code can also be downloaded in the Bootstrap library.

This button includes a download icon alongside some text: "Download it!".

Loving this book? Download it!

```
Loving this book? <a href="https://jupyterbook.org" class="btn btn-primary">
    <i class="fas fa-download"></i> Download it!
</a>
```

Banners and Announcements

Contents

A banner can be added to the top of a Jupyter Book:

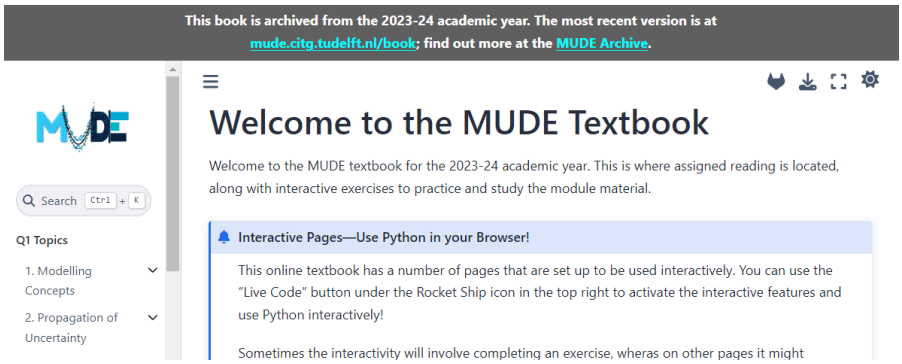


Fig. 93 Example of a banner in the MUDE book.

This is a commonly used feature, described in the Jupyter Book manual here. However, note that if you are using the Sphinx-Thebe interactive Python feature, or other features that customize the top part of a Jupyter Book website (for example, the download page buttons), the banner does not always work. In this case, the banner should be specified using the `announcement` option in the `html_theme_options` setting in the `_config.yml` file:

```
sphinx:
  config:
    ...
    html_theme_options:
      ...
      announcement : "This book is under construction ✍; it's continuously updated from
      ...
```

> **Tip**
>
> You should always enclose your announcement in quotes to make it a complete string. Use a different type of quote if your announcement itself contains them (for example, `"like 'this'"`). If you are using HTML elements, it is best to make your announcement a complete HTML tag, not Markdown with HTML inside.
>
> Markdown links do not work in the announcement, requiring you to use HTML!

Known Issue

Note that there seems to be a bug if you only use the `announcement` and `launch_buttons:thebe:true` option; including additional buttons seems to fix this issue (see Issue 65 in the MUDE book), for example:

```
sphinx:
  config:
    ...
    html_theme_options:
      repository_url: "https://github.com/TeachBooks/mechanics-BSc"
      use_repository_button: true
      use_issues_button : true
      announcement : "This book is under construction ✍; it's continuously updated from
      launch_buttons:
        thebe: true
```

Easy Editing Process

The following tools are created by TeachBooks to ease the process of making books. We acknowledge that making books is a difficult process; it requires advanced knowledge on CLI, python packages, html-files, data-storage and website hosting. The tools in this chapter are intended for those users who lack knowledge on these topics (which includes most of the TeachBooks' community members itself).

TeachBooks template

Contents

User types

This page is useful for user type 3, 4 and 5.

GitHub template

The template allows you to start your own TeachBook and hosting that TeachBook online without knowledge on Git, the Jupyter book package, python or anaconda. It doesn't elaborate on the collaborative functionalities of Git or how to edit the book. Please look at our manual (https://teachbooks.io/manual) to find more about that!

How to get started

How to use the template is demonstrated in the figure below, all steps are elaborated on in the following step-by-step tutorial.

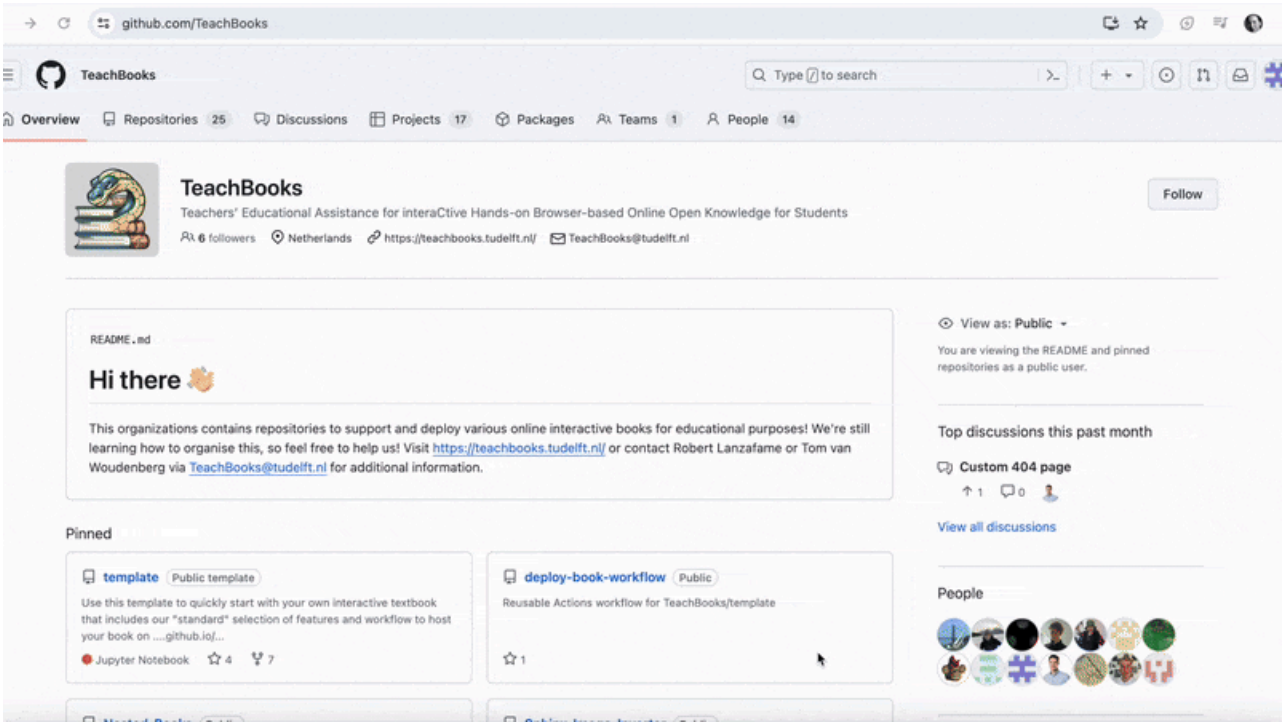
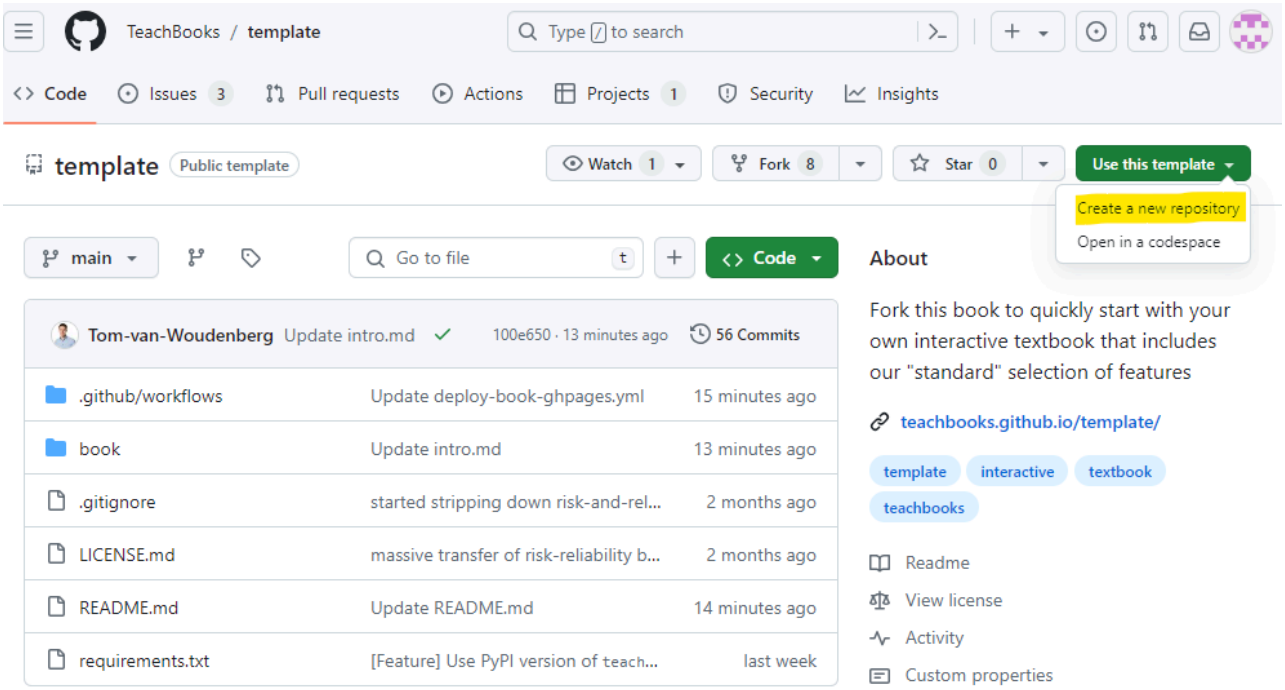
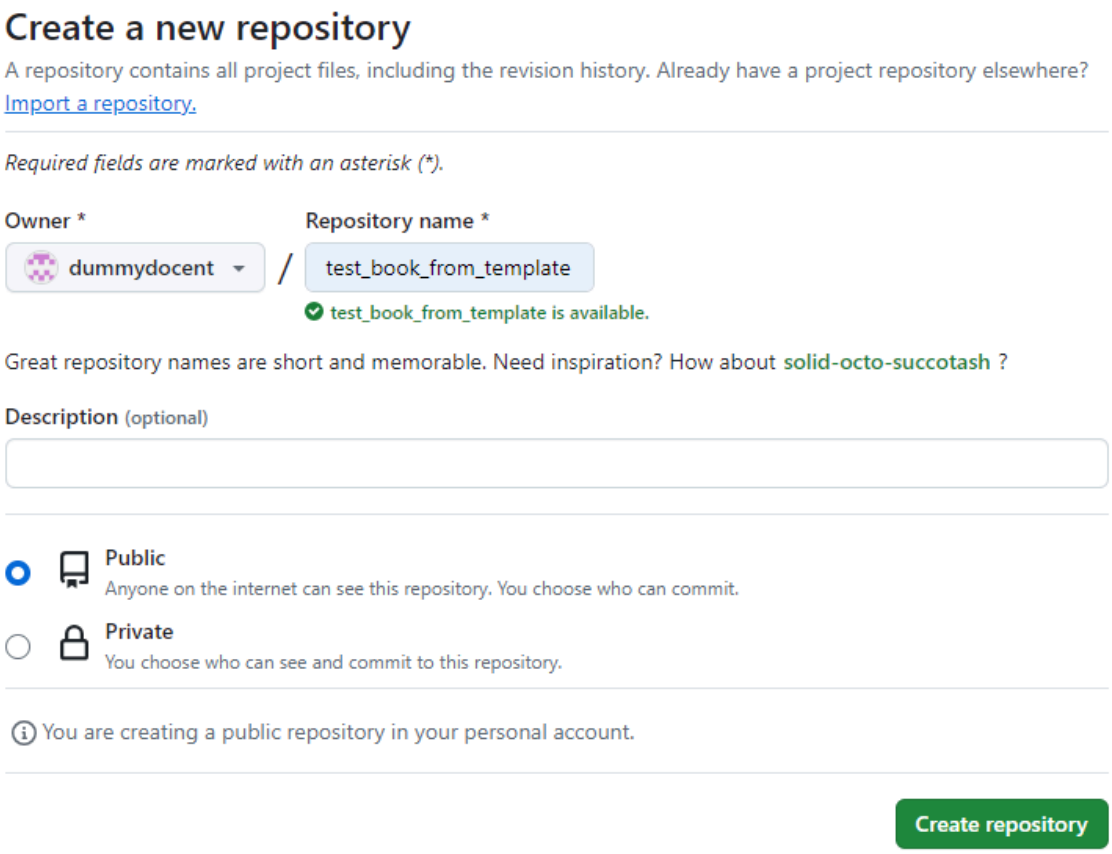


Fig. 94 Demonstration for a public repository, video available here

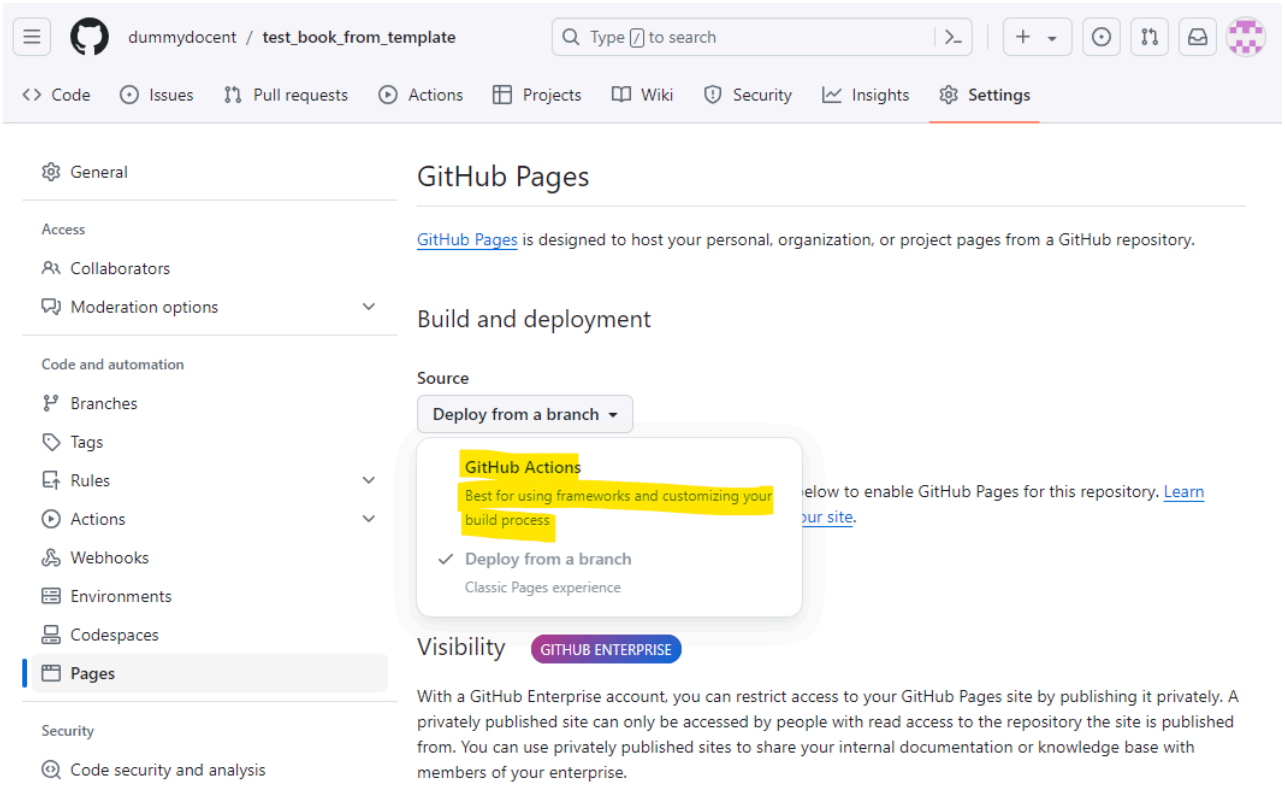
1. To get started making your TeachBook with our functionalities, use the template TeachBook as template:



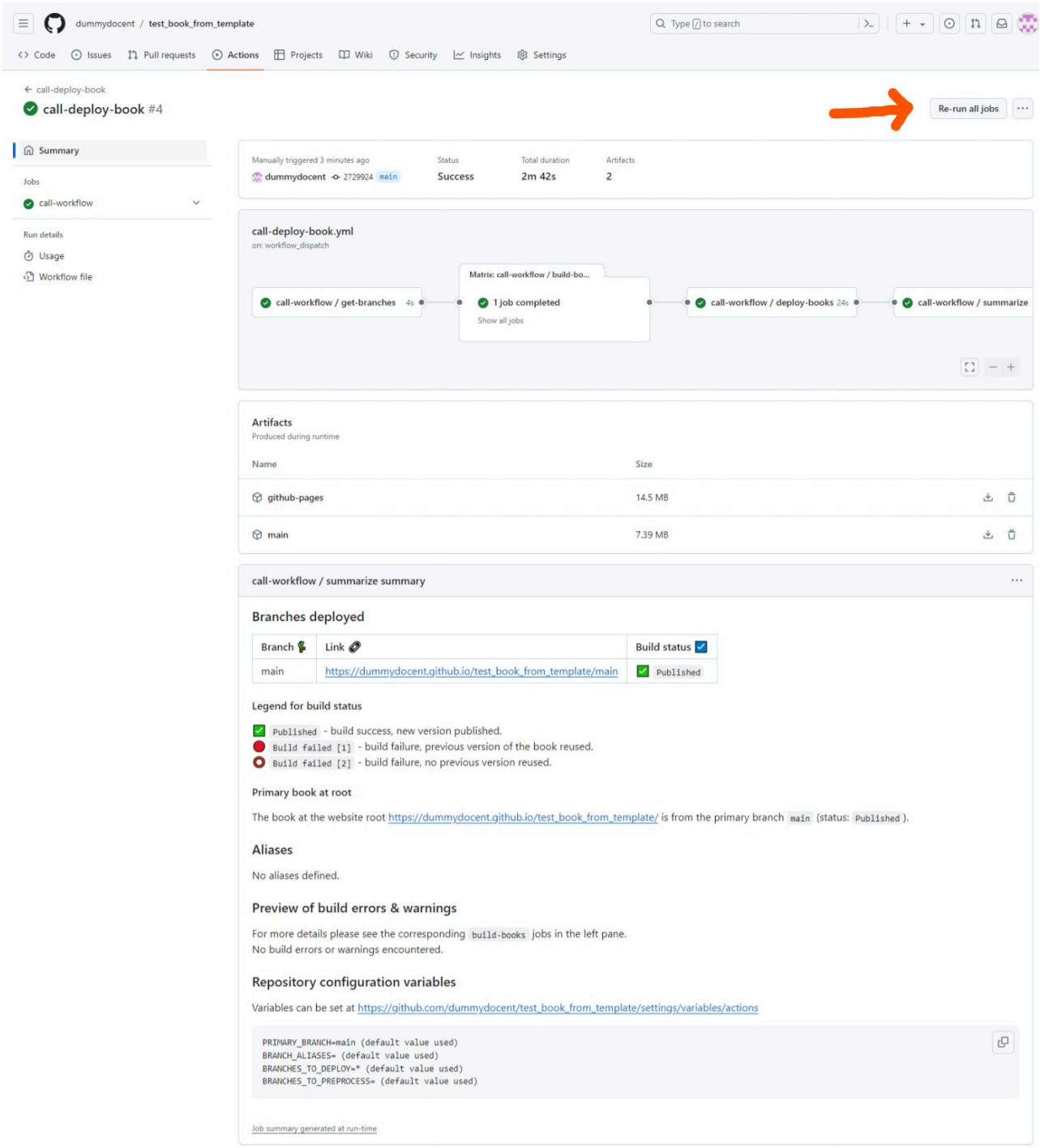
2. Fill in a repository name, this name will be used in the future url of your book:



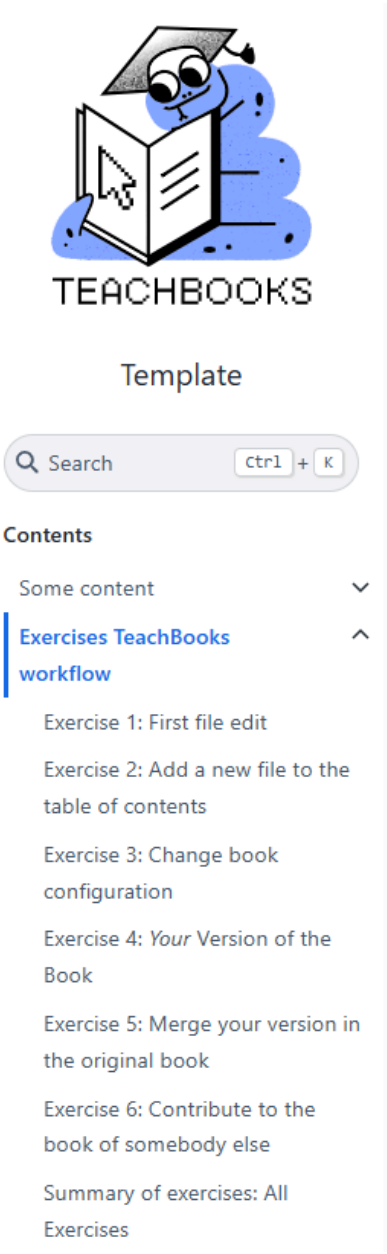
3. You can choose for `Private` only if you've GitHub Pro, GitHub Team, GitHub Enterprise Cloud, or GitHub Enterprise Server. Otherwise, you won't be able to publish your TeachBook online. Furthermore, it prevents people from contributing to your book, making your book essentially 'closed' instead of 'open'. Note that the built book website is always public.
4. You need to activate GitHub pages so that your website is published to the internet. As long as you don't do this your TeachBook is not published online. Actually, now that you've taken this template our workflow tries to publish it to GitHub pages, which you didn't have the chance to activate yet. That's why you probably received an email with 'call-deploy-book: Some jobs were not successful' and you see the failed job under ▶ `Actions` - `All workflows` - `call-deploy-book` - ✕ `Initial commit`. You can activate GitHub pages by setting the source for GitHub pages to GitHub Actions under ⚙ `Settings` - 🖵 `Pages` - `Build and deployment` - `Source` - `GitHub Actions`:



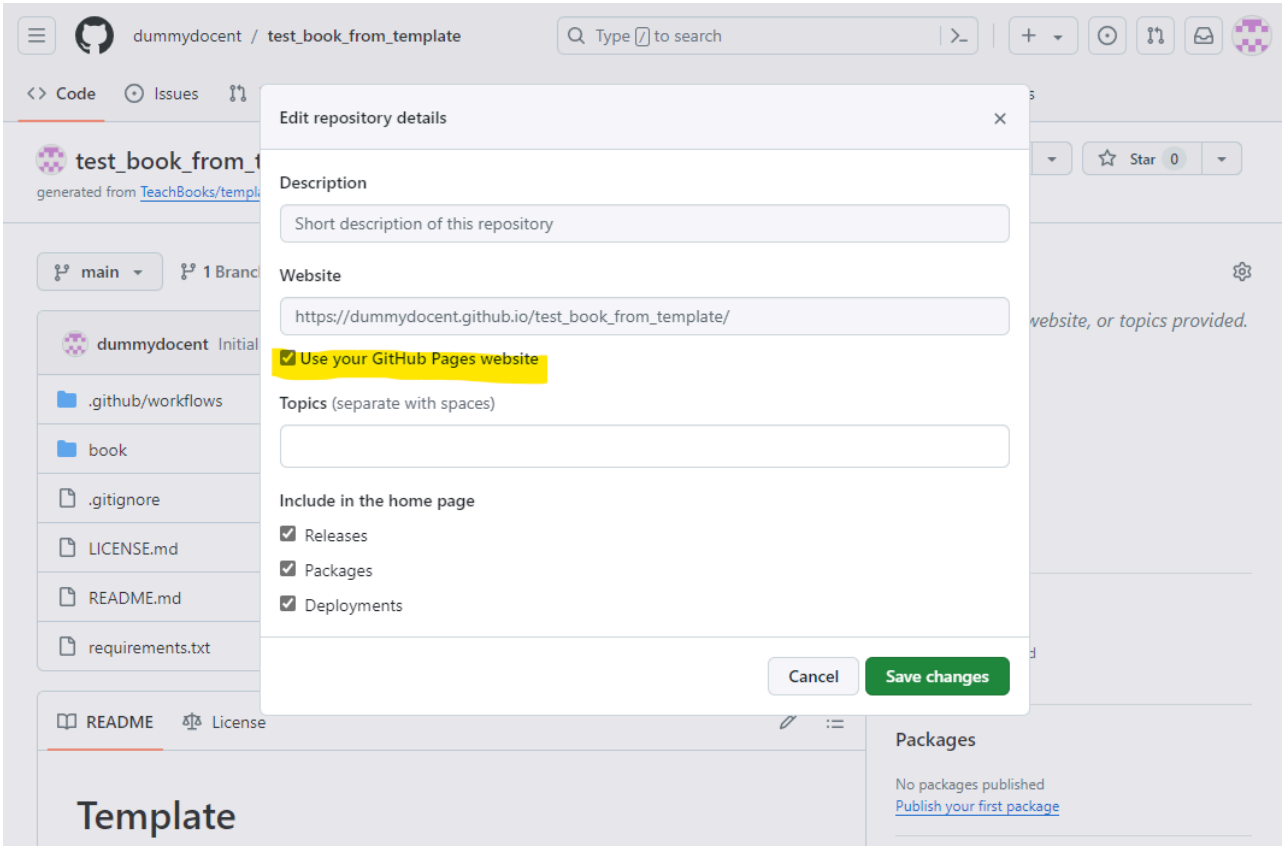
5. Make an edit to the TeachBook by editing and committing changes to one of the files in the `book/` subdirectory (available under <> `Code`). Now checkout the progress of the publishing workflow under ▶ `Actions` - `All workflows` - `call-deploy-book` - `<the most recent workflow run>`. Remember, the first commit which is there has failed because GitHub Pages wasn't activated at the time of `Initial commit`, you could also re-run that job if you don't want to make an edit. You can do so by running the workflow from ▶ `Actions` - `All workflows` - `call-deploy-book` - ✖ `Initial commit` - 🔄 `Re-run all jobs` - `Re-run jobs`:



6. When the workflow has finished, visit your build TeachBook at `https://<username or organiszation_name>.github.io/<repository_name>` (case sensitive). For our example it is https://dummydocent.github.io/test_book_from_template/ for the shown repository. These links are visible in the action's summary as well, as shown in the figure of step 4.
7. Want to get started directly? Your book contains a few exercises to get your started! Visit `https://<username or organiszation_name>.github.io/<repository_name>/exercises/exercises` (case sensitive) to get started with the first ones to get the basics of how to interact with your book on GitHub.



Additional tip: Set the repository website as your GitHub Pages website under <> `Code` - `About` - ⚙ - `Website` - `Use your GitHub Pages Website`



Features

- A github repository structure for making a Jupyter Book (`/book`)
- An empty TeachBook containing an intro page on root, an example markdown page, an example jupyter notebook page, an example references page. and an example credits page. (`/book/_toc.yml`, `/book/_config.yml`, `/book/credits.md`, `/book/intro.md`, `/book/references.md`, `/book/some_content/overview.md`, `/book/some_content/text_and_code.ipynb`)
- A file ready for adding references (`references.bib`, `/book/references.md`)
- An example favicon (web browser icon) (`/book/figures/favicon.ico`, `book/_config.yml`.)
- An example logo (`/book/figures/TUDelft_logo_rgb.png`, `/book/config.yml`)
- The configuration files set ready to make your Jupyter Notebooks pages work with live code using our sphinx-thebe extension and our recommended settings (`/book/config.yml`)
- An example of setting up preprocessing your table of contents to hide certain draft chapters for eg. students (`_toc.yml`)
- A file containing all the recommended software packages (`requirements.txt`)
- A file containing the recommended license CC BY 4.0 (`LICENSE.md`)
- Our GitHub workflow for publishing your TeachBook to GitHub Pages (`.github/workflow/call-deploy-book.yml`)

- A gitignore file containing standard python filetype to ignore (`.gitignore`)
- A readme containing information how to use the template, which can adjusted after using the template (`README.md`)

Contribute

This tool's repository is stored on GitHub. The `README.md` of the branch `manual_description` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

Draft-Release Workflow

Contents

Often it is necessary to prepare, review and edit materials in parallel to material that is currently being used by students, or another target audience. This workflow enables an author to easily maintain separate versions of a book without having to repeatedly comment out lines of a table of contents or page when building different versions. It is also easy to implement in CI/CD settings (it is already implemented in the {ref}Deploy Book Workflow `, which builds our TeachBooks).

> **Why is this useful?**
>
> Consider a case where you have been using two branches to develop and maintain a book: `release` (shared with students) and `draft` (shared with colleagues to review contents before release). You are probably also using other branches to develop contents, but rely on the `draft` branch as a way to collect the "finalized-but-not-yet-released" material. Things go smoothly as long as you add pages to the `draft` branch one at a time, then merge the branch into `release` as they are needed.
>
> But what happens when you have 2 pages: one is ready to share with students, but the other is still being reviewed by your colleague. The only way to get the proper page into release is to comment the undesired page in the ToC using a commit on `draft`, merge the commit into `release`, then uncomment the ToC in a new commit on `draft`. This is very tedious, and prone to error as there it is not straightforward to tell which pages are available in the `draft` versus `release` books!
>
> The Draft-Release workflow solves this problem by using commented tags to clearly denote which pages are available in the `draft` book and which are left out of the `release` book.

The workflow is enabled by a `teachbooks` CLI feature that identifies and removes any lines from the files `_config.yml` and `_toc.yml` file that are surrounded by `REMOVE-FROM-RELEASE` tags.

For example, pages in a developed book (e.g., `draft` branch) can be manually stripped out of the table of contents when a merge request from `draft` to `release` is made. This prevents the page being included in the released book. Pages marked with this feature are still visible in the `draft` book. Lines tagged in the configuration file `_config.yml` can be used in exactly the same manner. The tag is applied as follows:

```
format: jb-book
root: intro

parts:
  - caption: ...
    chapters:
    - file: ...
      ...
# START REMOVE-FROM-PUBLISH
    - file: files_to_remove
# END REMOVE-FROM-PUBLISH
```

There is no limit to the number of stripped sections, they can be sequential and indentation does not matter.

To invoke the tag and remove content during the book build process, use the following optional argument when building the book with the `teachbooks` package:

```
teachbooks build --publish book
```

Note that `teachbooks build book` would build a book without stripping the tagged lines, just as `jupyter-book build book` would. This is called “draft mode” and is the default behavior. You can recognize it in the stdout after running the `build` command:

```
TeachBooks: running build with strategy 'draft'
```

Installation

The package is currently available on PyPI and can be installed as follows:

```
pip install teachbooks
```

The first stable release (`v1.0.0`) is expected to be released in Spring 2025. Until then the package is very much useable but will be updated frequently. Therefore, get in the habit of updating the package regularly:

```
pip install --upgrade teachbooks
```

We recommend you install this in an environment that is specifically dedicated for building books. Python virtual environments are a good way to manage this and would be consistent with what is used by the GitHub servers via the Deploy Book Workflow. However, if you use non-Python tools to edit and check your book, a Conda environment may be a better choice.

The package is a CLI tool that primarily provides a wrapper around the Jupyter Book package which is used for pre- and postprocessing. In this case “wrapper” refers to the CLI usage: CLI commands generally invoke `jupyter-book` commands internally; the `jupyter-book` package is *not* distributed within the `teachbooks` package.

The source code and function of the package will eventually be documented on a Sphinx-built website (teachbooks.io/TeachBooks/), however, this is currently still under construction.

Releasing book online

Contents

We developed a workflow which builds your TeachBook in your repository for all branches and releases them online via GitHub Pages. In simplified terms, it automatically builds the book website based on updates to your repository, creates multiple instances of your book (defined by each branch) and provides the ability to customize the URL's at which each instance of the book can be accessed. This tool is designed specifically for educational contexts, for example, when you may want to preserve book versions from multiple academic years so that students are able to access it later. The TeachBooks Template uses this functionality for example.

The workflow `call-deploy-book.yml` calls the `deploy-book.yml` workflow, which builds a Jupyter Book at the calling repository for all branches, and deploys them via GitHub Pages. It is currently configured to create a Jupyter Book using the TeachBooks Python package (i.e., `teachbooks build book`), although this may be adapted in the future to make it easier to use in other applications (e.g., to build books with other software or any static website in general).

The workflow has the following features:

- Releasing of your TeachBook-repository (built with Jupyter Book) to GitHub Pages
- Ability to release both private (GitHub Pro, GitHub Team, GitHub Enterprise Cloud, or GitHub Enterprise Server required) and public (GitHub Free is enough) repositories.
  GitHub Teams is free for teachers as described in the GitHub documentation.
  If you have an organization for your TeachBook on GitHub, link your GitHub team rights to your organization as described on the GitHub website.
- Releasing all or a selection of branches, allowing to build a draft version of the TeachBook online which reduces the need for local builds of the book
- Provides a summary describing where the TeachBook is released, errors in the build process per branch and how the release step is configured
- Caching of already built books so that it can be partially reused when another branch is released or the next build contains critical errors
- Caching of python environment to speed up the workflow
- Allowing use of submodules within your book
- Customizable trigger for the workflow itself
- Optionally preprocess branches using the `teachbooks` package (e.g., Draft-Release Worklflow).
- Converting branch-names to well-defined URLs

- Customizable settings on where the books should be deployed including alias for branch-names and selection of one branch to be deployed on root. The workflow will gives warnings if these settings are ill-defined or conflicting. Although aliases are generally not allowed by GitHub Pages, it seems you can use one alias, but not more.
- Customizable behavior of book URL root directory, either by redirecting the root to one of the branches or by copying or moving one of the branches to root.
- Adds an 'archived'-banner to old branches / branches of previous years.

How to start using this workflow

As previously mentioned, this workflow is used in `TeachBooks/template`. Feel free to use it for your TeachBook as well:

1. Add `teachbooks` to your `requirements.txt` file in your root folder
2. Move all the TeachBook files (including `_config.yml` and `_toc.yml`) to a subdirectory `book/`.
3. Copy the `call-deploy-book.yml` workflow to the `/.github/workflows` folder in your repository.
4. Set source for GitHub pages to GitHub Actions under ⚙ `Settings` - 🗔 `Pages` - `Build and deployment` - `Source` - `GitHub Actions` (as long as you don't do this the workflow deploys all branches which build successfully).
5. Trigger the workflow by making an edit to the TeachBook by editing and committing changes to one of the files in the `book/` subdirectory (available under <> `Code`) or manually activating the workflow under ⏵ `Actions` - `All workflows` - `call-deploy-book` - `Run workflow` - `Use workflow from branch: <the branch you did step 1, 2 and 3 in>` - `Run workflow` (this workflow).
6. Now checkout the progress and summary of the releasing workflow under ⏵ `Actions` - `All workflows` - `call-deploy-book` - `<the most recent workflow run>`.

Customize the workflow: TeachBook releasing settings

You can adapt the behaviour by setting repository variables as explained here or using the VS Code Extension GitHub Actions. Define the following repository variables:

- `PRIMARY_BRANCH` which is set to `main` whenever it's not defined in the repository variables.
  - This sets the branch or alias (when using 'redirect' for `BEHAVIOR_PRIMARY`) which is shown on root.
  - It is advised to show either your default branch on root, or the branch shared with your primary/active book audience (e.g., your current students).
- `BEHAVIOR_PRIMARY` which is set to `redirect` whenever it's not defined in the repository variables.
  - This indicates whether to copy the PRIMARY_BRANCH to root ('copy'), move the PRIMARY_BRANCH to root ('move') or redirect from root to the PRIMARY_BRANCH ('redirect')
  - Advised to use 'redirect' if you expect to archive a version in the future so that the URL doesn't change for the reader (e.g., to preserve URL containing the current academic year shared with students).
  - Use copy or move when you only expect readers to use the root URL. Move is useful to remove unnecessary build artifacts and if you don't need to visit the URL containing the branch or alias name.
- `BRANCH_ALIASES` which is set to ` `(just a space) whenever it's not defined in the repository variables.
  - This defines an alias (custom URL) for a branch
  - Variables should be a space-separated list of branch names, e.g. 'alias:really-long-branch-name`
  - If no alias is wanted, `BRANCH_ALIASES` may be set to ` `(just a space)
- `BRANCHES_TO_DEPLOY` which is set to `*` (all branches) whenever it's not defined in the repository variables.
  - This defines the branches to deploy.
  - It should be a space-separated list of branch names, e.g. 'main second third'.
- `BRANCHES_TO_PREPROCESS` which is to to ` `(just a space = no branch) whenever it's not defined in the repository variables

    - This defines the branches to preprocess with the `TeachBooks` package, which removed book-pages and config lines defined with `# START REMOVE FROM RELEASE` and `# END REMOVE FROM RELEASE`
    - It should be a space-separated list of branch names, e.g. 'main second third'.
    - If no preprocessing is required, `BRANCH_TO_PREPROCESS` may be set to ' ' (space).
- `BRANCHES_ARCHIVED` which is set to ` `(space, no branch) whenever it's not defined in the repository variables
    - This adds a banner to the released branch: You are viewing an archived version of the book. Click here for the latest version.
    - It should be a space-separate list of branch names, e.g. 'main second third'.

In `call-deploy-book.yml` itself you can specify the trigger for this workflow. By default, a push to any branch triggers the workflow. You can limit the branches or subdirectories.

Common Usage Examples

Relevant use cases are explained here, along with an explanation for how to set up the workflow accordingly. Note that it is not required to set your `PRIMARY_BRANCH` to the default branch of your GitHub repository; this is a choice that is determined by what version of the source code you want visitors to see (i.e., work in progress, or the most recent "complete" or "released" version of a book).

Books with active users of different versions (academic years)

Consider a case where each academic year you would like to create a new book for your students. However, you need to ensure that students from previous years can still access "their" version of the book.

Assume for this example that we are working in the "my_organization" GitHub Organization in repository "my_book" and that the current academic year is 2025, and that we have one or more books in our repository from previous years. The desired URL structure is thus:

- students from this year use the book at `my_organization.github.io/my_book/2025/`
- students from last year use the book at `my_organization.github.io/my_book/2024/`
- visitors to `my_organization.github.io/my_book/` will automatically be redirected to the book from the current year

To create this behavior, do the following:

- Create a branch for each year: a logical name would have format `YYYY`, (technically it can be anything, as the URL can be set to `YYYY` with `BRANCH_ALIASES`).
- Set `PRIMARY_BRANCH` to the branch for the current academic year (e.g., `2025`)
- Set `BEHAVIOR_PRIMARY` to `redirect` (default)

Alternative without redirect to current year

One possible modification to this setup would be if you are progressively releasing content to your current students (e.g., `2025`) but you wanted visitors to `my_organization.github.io/my_book/` to see a complete version of your book. Assuming that the ideal version for such visitors is the completed book from the previous year (`2024`), you could do this:

- Set `PRIMARY_BRANCH` to `2024`
- Set `BEHAVIOR_PRIMARY` to `copy`

As your current students may then accidentally go to the older version of the book, we recommend you use a banner to indicate to readers that there is a (partial) new version available, and advise them where to find it. You can add a banner using this workflow (`BRANCHES_ARCHIVED`) or the standard Jupyter Book banner feature.

Books with active editors working in parallel

Consider a case where several authors are working on material for a book and you would like to be able to see the work of any author at any time. You also have a branch that is used to share finalized material with your readers (we call this a "release" branch) and a branch that is used to collect and review the work of all authors before releasing it (we call this a "draft" branch).

Assume for this example that we are working in the "my_organization" GitHub Organization in repository "my_book." The released book is on branch `release` the draft book is on branch `draft`; author branches are `author_1`, `author_2`, etc. The desired URL structure is thus:

- official (released) version of the book is at `my_organization.github.io/my_book/`
- draft version of the book is at `my_organization.github.io/my_book/draft/`
- the work of each author can be found at `my_organization.github.io/my_book/author_1/`, etc.

To create this behavior, do the following:

- Create a branch for each author as well as `release` and `draft` branches
- Set `draft` to the default branch
- Set `PRIMARY_BRANCH` to `release`
- Set `BEHAVIOR_PRIMARY` to `copy`

Note that in this situation we strongly recommend using the Draft-Release workflow that is incorporated in the TeachBooks Python package, which allows you to restrict specific content from appearing in the `release` book that is already incorporated in the `draft` book. Find out more in the TeachBooks Manual. This is useful, for example, if you would like to review many chapters, but release only one at a time, or use a banner in the draft book (e.g., via `_config.yml`) but not the released book.

Why make the draft branch default?

What if you want to make sure the source code for the released book is visible in the repository, rather than the draft version (e.g., you want to make sure external visitors see this version of the material)? You can do this by making `release` your default branch instead of `draft`. However, this also means that Pull Requests will be default be made into the `release` branch instead of `draft`, which may alter the working method of your team in an undesirable way (e.g., not being able to use a `draft` branch, accidentally releasing material before checking on `draft`, or having to manually adjust every PR that is created by GitHub to merge into `draft` instead of `release`).

Private submodules

In case your book includes private submodules, you'll need to create a Personal Access Token (classic) with at least the scope `repo` as described in the github documentation, and add this token with the name `GH_PAT` as a `Repository secret` or `Organization secret` (⚙ `Settings` > ✱ `Secrets and variables` > `Actions` > `Repository secrets` or `Organization secrets`.) Furthermore, manually copy the workflow (.github/workflows/deploy-book.yml) in your own repository as the authorization doesn't work with the called workflow.

Additional GitHub settings

We advise you to enable two options in the general repository setting regarding pull requests in GitHub:

- Enable `Always suggest updating pull request branches`, suggesting a merge from the default branch into any separate branch before merging into main.

- Enable `Automatically delete head branches` to delete branches after they are merged (you'll still be able to restore those).

View the workflow progress and summary

Whenever the workflow is triggered, its progress and a summary can be seen under the ▶ `Actions` - `All workflows` - `call-deploy-book` in GitHub! It shows you a descriptive summary:

- Ill-defined repository configuration variables (in `Annotations`)
- Which branches are released and where (`https://<username/organization_name>.github.io/<repository_name>` (case sensitive)) including which branch is released on the website root and the applied alias
- Errors in the build process
- How the repository variables are defined during the build.

Here's an example for a summary for the template book:

> ## Branches deployed
>
> | **Branch** 🌿 | **Link** 🔗 | **Build status** ☑️ |
> | --- | --- | --- |
> | main | https://teachbooks.github.io/template/main | ✅ `Released` |
> | version2 | https://teachbooks.github.io/template/version2 | 🔴 `Build failed [1]` |
> | version3 | https://teachbooks.github.io/template/version3 | ⭕ `Build failed [2]` |
>
> ### Legend for build status
>
> ✅ `Released` - build success, new version released.
>
> 🔴 `Build failed [1]` - build failure, previous version of the book reused.
>
> ⭕ `Build failed [2]` - build failure, no previous version reused.
>
> ### Primary book at root
>
> The book at the website root https://teachbooks.github.io/template/ redirects to the primary branch `main` (status: ✅ `Released`).
>
> ## Aliases
>
> | **Alias** ➡️ | **Target** 🎯 | **Link** 🔗 | **Build status** ☑️ |
> | --- | --- | --- | --- |
> | draft | main | https://teachbooks.github.io/template/draft | ✅ `Released` |

Preview of build errors & warnings

For more details please see the corresponding `build-books` jobs in the left pane.

On branch `version2`:

```
[91m/home/runner/work/template/template/book/some_content/overview.md:5: WARNING: Non-consecutive header level inc
```

On branch `version3`:

```
/home/runner/work/_temp/ff8c8325-8d8b-4c0b-a2b2-32d2169c55bc.sh: line 8: teachbooks: command not found
```

Repository configuration variables

Variables can be set at TeachBooks/template

```
PRIMARY_BRANCH=main (default value used)
BRANCH_ALIASES=draft:main
BRANCHES_TO_DEPLOY=* (default value used)
BRANCHES_TO_PREPROCESS=main
BEHAVIOR_PRIMARY=redirect (default value used)
BRANCHES_ARCHIVED= (default value used)
```

Book Build

The Jupyter Book is built using the TeachBooks Python package `teachbooks`, which is a wrapper around the `jupyter-book` command line tool that makes it easier to customize the build and deploy workflow. The package is documented at teachbooks.readthedocs.io/ for those interested in the package implementation and API; for those interested in a higher level overview and how-to, see the package overview page in the TeachBooks Manual.

By default the book is built using the following command:

```
teachbooks build book/
```

The Draft-Release workflow allows tagged lines of code to be removed from `yml` files (for example, if some pages of a book must be removed from a specific version). This is activated in the Deploy Book Workflow using the `BRANCHES_TO_PREPROCESS` variable, which will then execute the following command:

```
teachbooks build --release book/
```

More information about the Draft-Release workflow can be found here.

Caching

The GitHub Action Cache is used in the Deploy Book Workflow (DBW) to save time when building the book. This is accomplished by caching two sets of files for each branch: 1) the Python virtual environment, and 2) the build artifact (the HTML files forming each book website).

The reason for doing this stems from the primary purpose of the Deploy Book Workflow, which builds many versions of a book based on each branch, as well as creating a Python virtual environment from the `requirements.txt` file in the repository. A unique environment for each branch is necessary to test changes to book dependencies and configuration in an isolated way, but can dramatically increase the time required to build all versions of the book when the DBW runs. In addition, it also takes time to build the book itself. For this reason, both of these sets of files are cached (although at the moment, most of the time savings is primarily caching the build artifact).

This works by hashing a specific set of files and including that in the filename of the cache. Every time the workflow is triggered, a new hashed filename is created and compared to the list of existing caches to see if one with the same name exists; if so, the cache is reused. If not, a new environment and/or build artifact is constructed and a new cache is made.

Cached files can be found in GitHub under the Actions tab, then looking for the "Caches" section under the "Management" pane on the left hand side of the screen. For example, this link shows the caches for the TeachBooks Manual. If needed, caches can be deleted manually from this page. A cache will expire if unused for longer than 1 week or if the maximum allowed disk space for all cached files is exceeded (both are GitHub policies).

Files created during a GitHub Action are called artifacts. Although this is *not* the same thing as a cache, the files in a specific cache or artifact are identical. Artifacts have their own expiration period, which is possible to customize in the repository settings (default is 90 days). Unexpired artifacts can be viewed at the bottom of any workflow run summary that is available on the Actions tab of a GitHub repository.

To add even more confusion to the situation, if an artifact built during an Action is a set of HTML files (a website) that are used to create a GitHub Pages website, the files served when visiting the URL for that are stored on a webserver; they are *not* the artifact files, but are identical. Fortunately webserver files are preserved indefinitely (or until GitHub changes this policy), meaning that even though your cache or artifact may be deleted, the website will still remain active.

The cache is a key component of the DBW and leads to several considerations for the building and maintenance of a book, which are explained below, after the criteria for each cache type are described.

Cached Environment

The DBW uses Python virtual environments (`python -m venv .venv`) and `python -m pip install -r requirements.txt` to install packages and create the book building environment. The entire `.venv` directory is preserved in the cache with a file name beginning with `venv-...`. The week number is also included in the file hash, which ensures that a new hash is created at least once per week.

Once created during a successful build action, an existing cached environment will be found and reused unless two specific criteria are met: 1) a book build is required (replacing the cached build artifact), *and* 2) the `requirements.txt` file is changed. Requirements for a "book build" are described in the next section.

Note that any change to the `requirements.txt` file will trigger creation a new environment, not necessarily one that specifies the version number of a package.

Cached Build Artifacts

Build artifacts are the HTML files that define the website (i.e., the book) and are typically located in the subdirectory `book/_build/html` of a repository once the book is built. Note that these files are typically only visible if you are building the book locally, as they are not committed to the repository by default (i.e., `.gitignore`). For users only using the DBW and the GitHub browswer editor, downloading the build cache is the easiest way to view these files, which have filenames beginnig with `html-build-<branch-name>-...`.

Once created during a successful build action, an existing cached build artifact will be found and reused unless *any* of three specific criteria are met: 1) a change is made to a file in `book/`, 2) the `requirements.txt` file is changed, or 3) the status of a

branch as archive or preprocess has changed (`BRANCHES_TO_PREPROCESS` of `BRANCHES_ARCHIVED`).

Immediately after a successful build action, the book website files exist in three places: cache, artifact and on a GitHub webserver. However, as described at the beginning of this section, the cache will be automatically deleted after 1 week, if unused, and the artifact will be deleted based on the setting in your repository (90 days by default).

Effect of Caching on Book Workflow

Implementation of the DBW has several characteristics that should be noted which could help understand certain behaviors of the book build and website deployment process. These points may be useful to consider if you are experiencing undesired behavior in your Actions builds and/or your actual book websites.

- Remember that the DBW action will only run if a commit is made that changes `requirements.txt` or contents of `book/`
- When first creating a repository, the action may not run, or may run and fail; occasionally it is needed to make a new commit to get the workflow to run for the first time (remember to modify something in `book/`) or you may try to re-run the job from the Actions tab
- when using multiple branches, only the branch that is edited will be updated
- the website for each branch may be built with a different Python environments (different package versions)

Unlike the virtual environments (next subsection), caches of the book build do not influence subsequent builds of the book, as this artifact is replaced whenever a commit to a specific branch triggers a new build process. As described above, old cached build artifacts are used if the build process from a new commit fails, ensuring that the URL of a website does not return a 404 page.

Effect of Caching on Virtual Environment

In particular, note that older branches may have been built with cached environments that are different than those in a newer branch, leading to (undesired) differences in book appearance or functionality—often without the author being aware! This occurs if a package in `requirements.txt` is updated in the time between the creation of environments on different branches, as `pip` will use the newest version when downloading a package (the first time a venv cache is created in the DBW), but it will not always automatically update a package if `pip install -r requirements.txt` is used on an existing environment (this happens every time an existing cache is used!). This means that, when compared to the venv on a newer branch, a frequently used branch (where the cache has not expired) may retain old and out of date packages long after an updated version is available; however, since the week number is included in the venv file hash, this will last for a maximum of 1 week. A best practice to avoid this situation is to pin version numbers explicitly (e.g., `teachbooks==0.2.0`) if you want book builds to remain consistent. In addition, a feature like dependabot can be used to automatically notify you when an update is made; this is described in the TeachBooks Manual.

Contribute

This tool's repository is stored on GitHub. The `README.md` of the branch `manual_docs` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

To test changes to the deploy book workflow do the following:

1. Create a fork of the repository and make a commit to `deploy-book.yml`
2. Using a book repository (e.g., a book created from the TeachBooks Template), edit the `call-deploy-book..yml` file such that `TeachBooks/deploy-book-workflow/.github/workflows/deploy-book.yml@main` is replaced with the repository and

branch path to your modified file `deploy-book.yml`

3. Make a commit in the book repository that triggers the workflow and confirm that the job runs successfully.

Future improvements to the DBW may address the ability for a user to specify additional aspects of the build process, for example, the Python version or the specific book build commands (e.g., `teachbooks build` or `jupyter-book build`). If this is something that interests you, please create an Issue in the repository (perhaps with “feature request” in the title).

Build pull requests from forks

Contents

When writing a TeachBook it is useful to automatically build and view these changes online. This exactly why we designed and build the deploy-book-workflow tool, which also allows for multiple versions of your book to exist at any time with customizable URL's. However, it doesn't cover all use cases. Luckily, **Read the Docs** is a free tool that can easily be used for this purpose. It also provides a very easy way to view differences between versions on each page!

Pull Requests are a key tool for allowing anyone with a GitHub account to make a contribution to your work by forking your repository, adding commits, then sending them back to your repository via a Pull Request. Unfortunately you are not able to automatically build the book based on proposed changes in a pull request from a fork of your repository, as this requires the deploy-book-workflow in the fork to have GitHub Actions enabled to show the proposed changes in the book. Furthermore, when proposing your contribution in a pull request, you'd have to manually refer to your own built book (e.g., with a hyperlink). This process can be made easier by using Read the Docs, as it can automatically build a book based on changes proposed in a pull request, and includes a link to the book directly in the pull request page.

Read the Docs is not recommended for final versions of your book because of the advertisement in the free version. If you'd like to pay for it, it can replace the functionality of the deploy-book-workflow.

This tool works using Sphinx, which is the core engine for Jupyter Book; it is carried out using a Jupyter Book command to generate a Sphinx configuration file, for example:

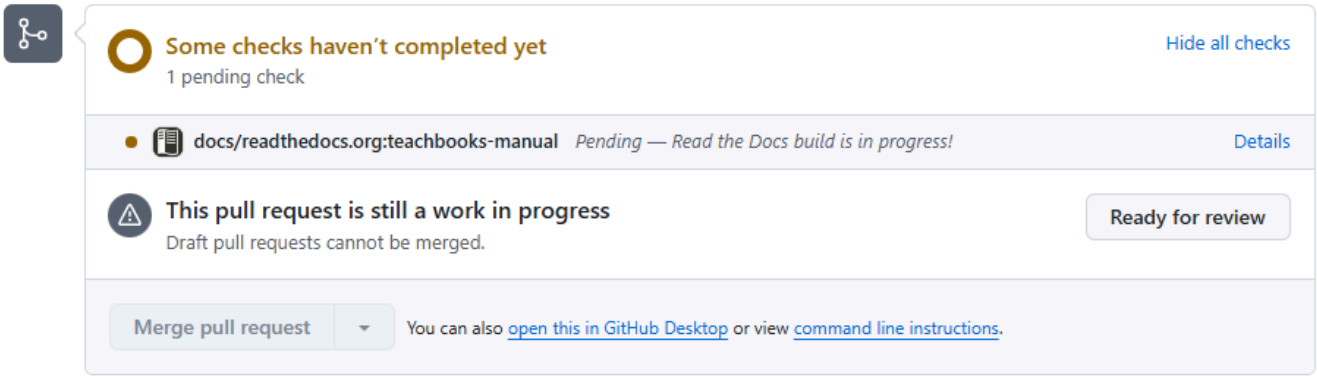
```
jupyter-book config sphinx book/
```

In theory this should work with the TeachBooks Python package `teachbooks` as well as local extensions (e.g., those in `book/_ext` like APA References), but we have not checked this yet. If you are interested, try setting it up as a PR related to this issue.

> **Tip**
>
> Once set up, this tool is only accessible via the Pull Request page for a repository in the 'Checks' part of the automated rule set box (illustrated below under 'Usage'). This is different from our deploy-book-workflow, which is accessed under the Actions tab of a GitHub repository.
>
> A Read the Docs account and admin privelages for a repository are the only requirements needed set this up; all visitors to the pull request page will be able to view the book, including the differencing feature.

Usage

After visiting the summary page of a pull request you will be able to see the following line (*make sure you click "show all checks" to see it!*):



Click 'Details' to see the logs of the build process.

Whenever the build is done, click 'Details' again to see the build book. Tip, click `d` or add `?readthedocs-diff=true` to the url to see the differences on the pages highlighted. Note that this differencing functionality is not perfect as it might indicate elements which are not changed intentionally. It is known to have issues visualizing changes associated with buttons, LaTeX and figures.

Example

As an example, Read the Docs is configured in this example book. Fork it and open a pull request to test it's functionality.

There's an example pull request with corresponding built book on Read the Docs.

Setting up

This tool is a specific automation rule that is enabled within the GitHub pull request ecosystem.

Add configuration file to your repository

Add a file `.readthedocs.yaml` to the root of your repository, containing the following content:

```yaml
version: 2

build:
  os: ubuntu-24.04
  tools:
    python: "3.13"
  jobs:
    pre_build:
      - "jupyter-book config sphinx book/"

python:
  install:
    - requirements: requirements.txt

sphinx:
  builder: html
  fail_on_warning: true
  configuration: book/conf.py
```

You might need to add `standard-imghdr` to your `requirements.txt` file if the build fails when you've configured it in Read the Docs (if the error tells you `ModuleNotFoundError: No module named 'imghdr'`)

Setup Read The Docs account

Setup an account at app.readthedocs.org/accounts/login/. We recommend using your GitHub account for authentication.

When authorizing Read the Docs for your GitHub account, you can also grant access by Read the Docs for an organization of which you are an owner, as well, which we also recommend.

Authorize access to organization

When you didn't do so in the previous step, or you'd like to grant access at a later moment, you can grant/revoke access by Read the Docs to your organization by following these menu items in settings: `personal GitHub Settings - Integrations - Applications - Authorized OAuth Apps - Read the Docs Community`.

Add project in Read The Docs

On your dashboard, add a new project. Find your GitHub repository (you might need to refresh your repositories if you've recently updated authorizations).

Enable Pull Request build

In the setting of your newest project, enable build of pull request by selecting the option in `Building - Pull request builds - Build pull requests for this project`.

Additional settings

The following settings are recommended:

- `Setup` - `Settings` - `URL versioning scheme` - `Multiple versions without translations (/<version>/<filename>)`, as translated books can be implemented with Multilingual book
- `Setup` - `Settings` - `Addons` - `Search` - disabled, as your books already contain a search function and the Read the Docs search functions requires additional setup
- `Setup` - `Settings` - `Addons` - `Link previews` - disabled, `Multiple versions without translations (/<version>/<filename>)`, as previews can be implemented with Rich hover over tips
- `Setup` - `Settings` - `Automation rules` - `Add rule`, to build all tagged versions of your book:
  - `Description`: 'Build all tags'
  - `Match`: `Custom Match`
  - `Custom match`: `.*`
  - `Version type`: `Tag`
  - `Action`: `Activate version`

Local Server to view interactive elements locally

Contents

Perhaps you have already noticed that when building a book and testing it locally on your computer, some features do not work the way you expect, or worse—do not work at all! This is because many interactive features rely on browser functionality to work properly, for example: Grasple/H5p iframe exercises, Sphinx-Thebe Python interactivity and HTML/Javascript elements. Although the website is *static* (i.e., there is no code running on a webserver that generates the content), modern web browsers have their own internal computing environments that run processes to deliver the rich content we desire in our books.

Technically what is needed to facilitate this during the editing and checking of your book locally is a **static web server** that can serve the files in the `./book/_build/html` directory of your book (this subdirectory is created as a result of the command `build book`). We refer to this as a **local server** because it is running on your local machine.

> **Why do you need a local server?**
>
> A Jupyter Book is really just a *static website*. This means that all book content *and interactivity* is available via the (static) files that are provided to you by a webserver when you visit a specific URL. After this initial request, all required files are available in your web browser, however, some features require the request and response protocol to work properly.

We recommend you use the **TeachBooks Python Package**, which has a `serve` command in the CLI tool to start and stop local servers. Find out more on the TeachBooks Package page.

This page presents just a few additional examples that are used by TeachBooks collaborators. There are many more options available out there, for example, this list illustrates many servers that can be activated with a single line of code!.

> **Tip**
>
> Remember that the browser is serving static files provided by the local server and does not always keep track when they are updated.
>
> You should reload the page if you are editing and rebuilding the book. You can try `CTRL+R` `ctrl`+`F5`. If this does not work, on Chrome try right-clicking somewhere on the page, select “Inspect”, open the “Network” tab, then reload with `CTRL+R`.

TeachBooks Server: `teachbooks serve`

If you have the TeachBooks Python package installed (`pip install teachbooks`), starting a server is as simple as:

```
teachbooks serve
```

See the TeachBooks Package page for additional guidance on using the CLI tool.

By default it assumes you have a book in `./book/`, are running the command from the root `./` of your repository and will serve `./book/_build/html`. If not, it will serve the repository root. You can also specify the directory to serve:

```
teachbooks serve path unconventional_book_dir/_build/html
```

> **Tip**
>
> Remember that the browser is serving static files provided by the local server and does not always keep track when they are updated.
>
> You should reload the page if you are editing and rebuilding the book. You can try `CTRL+R` `ctrl`+`F5`. If this does not work, on Chrome try right-clicking somewhere on the page, select "Inspect", open the "Network" tab, then reload with `CTRL+R`.

To stop the server, simply run

```
teachbooks serve stop
```

The CLI tool is set up to make the book edit and check workflow as easy as possible locally:

- you can rebuild the book without restarting the server.
- it reuses an existing server if already running.
- it prints the URL where the book is served after building the book to easily access it by clicking the link in standard output.
- the `teachbooks serve stop` command prevents Python processes running in the backgroudn and slowing down your computer (it saves the server state in a pickle file).

Installation of TeachBooks

The TeachBooks package is currently available on PyPI and can be installed as follows:

```
pip install teachbooks
```

The first stable release (`v1.0.0`) is expected to be released in Spring 2025. Until then the package is very much useable but will be updated frequently. Therefore, get in the habit of updating the package regularly:

```
pip install --upgrade teachbooks
```

We recommend you install this in an environment that is specifically dedicated for building books. Python virtual environments are a good way to manage this and would be consistent with what is used by the GitHub servers via the Deploy Book Workflow. However, if you use non-Python tools to edit and check your book, a Conda environment may be a better choice.

The package is a CLI tool that primarily provides a wrapper around the Jupyter Book package which is used for pre- and postprocessing. In this case "wrapper" refers to the CLI usage: CLI commands generally invoke `jupyter-book` commands internally; the `jupyter-book` package is *not* distributed within the `teachbooks` package.

The source code and function of the package will eventually be documented on a Sphinx-built website (teachbooks.io/TeachBooks/), however, this is currently still under construction.

Live Server: a VS Code Extension

A local server is easy to initialize directly in VS Code using the Live Server extension. Once installed, it can be activated with a "Go Live" button in the bottom right corner of VS Code. The extension will automatically serve index.html, so depending on your book build settings you may need to navigate to the correct directory/file manually using the URL address in the browser.

> **Install Live Serve Extension**
>
> You can install this extension by searching for `Live Server` (or even easier, extension ID `ritwickdey.LiveServer`)in the Visual Studio Code extensions marketplace.
>
> 

Python Server: `python -m http.server`

The Python standard library has a built-in server that can be activated from the command line. This is a simple way to serve a book locally

```
python -m http.server
```

If you don't mind typing a bit more, you can specify the port number and the directory to serve:

```
python -m http.server 8000 --directory book/_build/html
```

This is actually what the `teachbooks serve` command is using, but there we have added additional features to make the process more user-friendly for the book edit and check workflow.

Stopping the Python Server

It can be challenging to stop the Python server, especially when using Windows OS. Even though you may force end the process (e.g., with `CMD+C`), the server may still be running in the background.

For Mac and Linux users (or for Windows if you have Git Bash installed or another Unix-type terminal), you can try this:

```
pid=$(ps aux | grep http.server | head -n 1 | awk '{ print $2 }')
kill -9 $pid
```

Or this:

```
tskill python
```

A Windows-specific command for the CMD prompt or Powershell is:

```
TASKKILL /IM python.exe /F
```

Sharing content between books in table of contents

Contents

Introduction

When creating books, you might want to reuse material from other people or from other books you made. In some cases you might even want to have the exact same material into your book. You could do so by manually copying material over. However, whenever the source material is updated, you have to do that again. As an alternative, you can use the underlying git system to refer to the source file directly. This allows you to pick a specific version, or keep the most up-to-date version of it. This pages explains how to do so directly in the table of contents using the **External Content** (or **External TOC**) feature which is part of the TeachBooks Python package.

Previously, this book feature was implemented using submodules, but the implementation was more difficult to use. Despite this, submodules are still a widely used Git feature that can be very useful for book authors, so check out the submodules page to learn more, especially if the External TOC tool does not satisfy your needs. Submodules have a few additional features not (yet) implemented using the External TOC.

The External TOC features are incorporated in the teachbooks package starting with version `0.2.0`.

What does it do?

This functionality, part of the TeachBooks package allows you to refer to `.ipynb`, `.md` and `.rst` files stored on public repositories. These files are then handles as if they were normal files in your book, leading to a the file as a page in your built book. To prevent issues it validates that the external content has a permissive license, automatically combines any `reference.bib` files, and warns you if there's a mismatch in `requirements.txt` or with plugins in `_config.yml`.

Installation

To use this extenstion, follow these steps:

Step 1: Install the Package

Install the module `teachbooks` package using `pip`:

```
pip install teachbooks>=0.2.0
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
teachbooks>=0.2.0
```

Usage

To use the functionality, take the URL of a file and add it to your `_toc.yml` as follows:

```
format: jb-book
root: .

parts:
  - caption: ...
    chapters:
    - file: ...
    .
    .
    .
    - external: <link to GitHub / GitLab source .md or .ipynb file>
    .
    .
    .
```

The link can be pointing to a file on a branch (which will take the most recent version of that file on the branch) or a commit or tag (which will take a specific version of the file). For example:

- `external: https://github.com/TeachBooks/manual/blob/release/book/intro.md` will take the most recent version of the intro page of this book on the release branch
- `external: https://github.com/TeachBooks/manual/blob/06d67e8a0110c94d9147ce07090656dedc7d0e64/README.md` will take the file during the state of a specific commit (06d67e8).
- `external: https://github.com/TeachBooks/manual/blob/v1.1.1/README.md` will take the file during the state of a specific tag (v1.1.1)

If you're building your book online in GitHub, the deploy-book-workflow will deal with these files during the `teachbooks build` command. If you want to build the book locally, run the command `teachbooks build`.

If the source file is from another book, the contents of the `requirements.txt`, `_config.yml` are checked for any missing/incompatible entries. If this leads to compatibility issue, you'll be warned during the build which will help you solve these dependencies or plugins manually by updating your main book's `_config.yml` and `requirements.txt` Additionally, any `references.bib` files are merged into your main book's `references.bib` file, and the licenses of the external content are validated to allow for re-use. External content without a permissive license will result in an error to try to prevent any accidental copyright infringement.

Contribute

This tool's repository is stored on GitHub. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository.

Share content between books using submodules

Contents

Git Workflow

When creating books, you might want to reuse material from other people or from other books you made. In some cases you might even want to have the exact same material into your book. You could do so by manually copying material over. However, whenever the source material is updated, you have to do that again. As an alternative, you can use the underlying git system to refer to the source file directly. This allows you to pick a specific version, or keep the most up-to-date version of it. This pages explains how to do so using 'git submodules'

> **Tip**
>
> This feature is now considered deprecated by the TeachBooks Development Team because we have developed an easier way to incorporate content from other source via the "external content" module in the `teachbooks` Python Package. You can find out more by visiting the TeachBooks Manual or GitHub repository. Note, however, that if you are comfortable with the steeper learning curve, the submodules feature of Git is still useful for some use cases, as it allows one to embed pages in a book which:
>
> - include local images referenced with raw-HTML code and relative filepath references,
> - refer to content in the `_static` folder of the other book (i.e., relative filepath references after book build),
> - are part of a private book (private repository),
> - *automatically* notify you upon changes in the source book using Github's "Dependabot", mentioned below.
>
> More generally, you should consider submodules if you want to have all of the files in a submodule book repository available in the main book repository (especially during and after the book build process); if you would like to use the dependabot feature to automatically update your book every time a commit is made in the submodule repository; or, if you prefer to update content based on a specific commit hash rather than a tag or branch. In fact, submodules is how we keep the pages in this manual up to date for each of our TeachBooks Tools!

Adding external content to your book

If you want to add the repository of an external book to the repository of your book you can do so by importing the other repository using Git Submodules. Your book is then called the parent book. This will basically nest the external repository in

the parent repository, and it will appear as if we've manually copied the entire repository into the parent repository. I will use this repository as an example parent repository, and I'm going to add parts of an old MUDE book.

First, let's define the location where the external book should live. Good practice is to put it in `book/external` to highlight the fact that the content is in fact part of an external book. So the first step is to create the subdirectory `book/external/` in your repository. If you'd like to do that with the command line interface (CLI) you can do so as follows:

```
mkdir -p book/external
```

Now we can add the external book in this folder. You'll need to use the CLI for that! (open the CLI by opening Git Bash or click `Repository` - `Open in command prompt` in GitHub Desktop):

```
cd book/external
git submodule add -b <branch> <https-link>
```

where

- `<branch>` should be the branch you want of the external book/repository. If you only want the default branch, omit the part `-b <branch>` from the command.
- `<https-link>` is the link to the external book/repository, for example `https://github.com/tudelft-citg/mude.git`.

You can see that the `book/external` directory now contains a directory with the name of the external repository (`MUDE` for example), so the result looks equivalent to simply running a `git clone` inside `book/external`, however what is important to note here is that the contents of `book/external/<external repository>` are not part of the parent repository. Instead, `book/external/<external repository>` is a fully functional Git Repository itself. This means that you can make changes to the external book, from inside the parent book.

If you did not add the submodule with an explicit branch, you can still do this by adding this to the `.submodules` file that has been created. In our example, we see the lines

```
[submodule "book/external/MUDE"]
        path = book/external/MUDE
        url = https://github.com/TUDelft-CITG/MUDE
```

Changes these to

```
[submodule "book/external/MUDE"]
        path = book/external/MUDE
        url = https://github.com/TUDelft-CITG/MUDE
        branch = <branch>
```

with again `<branch>` replaced by the preferred branch.

After the `git submodule` command, you can make a commit to your parent repositery:

... using CLI | **... using GitHub Desktop**

```
git commit -m "Add external book"
```

Now, you can add sections of the external book to the table of contents of your parent book (`_toc.yml`):

```
chapters:
- file: external/MUDE/book/intro.md
```

In the above line, I have added the introduction of the MUDE book to my book.

You might want the title of your book page in the table of contents to be different than the title provided as the first header in your nested file. You can adapt the title by specifying it in `_toc.yml`:

```
- file: <somefile>
  title: <Some custom title to show in the table of contents>
```

Editing

If you want to make an edit to the content of an external repository, which is a submodule of your parent repository (meaning it's nested), you'll need to make changes to the external repository so that the parent repository has a commit to point to when 'updating' its content. You can do this from within the parent repository! We'll break down the steps:

... using CLI | **... using GitHub Desktop** | **... using GitHub Dev with codespaces online**

1. Navigate to the parent repository

Open your terminal, for example Git Bash or the integrated terminal in VS Code and use the `cd` command followed by the full path of your parent repository that you have previously cloned to your laptop.

```
cd /<path>/<parent-repo>
```

Alternatively, you can locate the submodule in your files, hover your mouse above the path and `right click` - `copy adress as text`. Then you can paste the adress in between quotation marks. (This works only on windows)

```
cd "C:\<path>\<parent-repo>"
```

Once you are in the right folder, git bash will indicate the branch you are in in blue brackets.

2. Update the submodule

In case someone else has edited the content of the submodule you want to make sure that you ave pulled all recent changes.

```
git submodule update
```

3. Navigate to the submodule

```
cd /<path>/<parent-repo>/external/<submodule>
```

Or

```
cd "C:\<path>\<parent-repo>\external\<submodule>"
```

4. Checkout a new branch (optional)

It is good practice to create a new branch for every set of changes you make to the conent.

```
git checkout -b <new-branch>
```

5. Make changes, stage, and commit them in the submodule

Now you can make changes to your file. To open the file in VS Code type:

```
code <file-name.ext>
```

Save the file in VS Code and return to the terminal. To stage all changes type:

```
git add .
```

If you only want to stage one file type:

```
git add <file-name.ext>
```

Next commit the changes:

```
git commit -m "Commit message"
```

6. Push the submodule changes to its remote repository

Next you'll need to push the changes to the external repository. The syntax you need to follow to push is `<repository> <branch>`. The repository name will most likely be origin which refers to the cloned repository you are in.

```
git push origin <branch-name>
```

7. Return to the parent repository

```
cd /<path>/<parent-repo>
```

Or

```
cd "C:\<path>\<parent-repo>"
```

8. Stage the submodule update in the parent repo and commit

Now we have to 'update' the parent repository with the changes pushed to the external repository (submodule). First, stage the submodule update. Then commit the change.

```
git add book/external/<submodule>
git commit -m "Updated submodule"
```

9. Push the parent repository changes

Finally push the update to the remote repository.

```
git push origin <branch-name>
```

Cloning with submodules

If you're cloning a repository that features submodules, the directories of the submodules will not be populated by default. To fix that, you need to do a recursive clone (i.e., clone the parent repository, as well as the submodules):

```
git clone --recurse-submodules <link to parent repository>
```

The external book is updated

When you add the external book as a submodule to your repository, Git will pin its version. When the external book is updated, you'll need to manually pull the updates to the parent book or use the automatic GitHub Dependabot.

automatic using GitHub Dependabot | **manual using CLI** | **manual using GitHub Desktop**

It is possible to set up Github in such a way that periodically and on demand the submodules in the *default* branch of the parent repository. To enable this, perform the following steps:

1. Go to your repository on Github.
2. Choose **Settings**.
3. Choose **Code security**.
4. Choose **Enable** behind *Dependabot version updates*.
5. In opened file editor, edit the code to resemble below code and select "Commit changes..."

```
# To get started with Dependabot version updates, you'll need to specify which
# package ecosystems to update and where the package manifests are located.
# Please see the documentation for all configuration options:
# https://docs.github.com/code-security/dependabot/dependabot-version-updates/configuration-options-for-the-dependabo

version: 2
updates:
  - package-ecosystem: "gitsubmodule" # See documentation for possible values
    directory: "/" # Location of package manifests
    schedule:
      interval: "weekly"
      day: "sunday"
      time: "23:59"
```

This will check every sunday around midnight (UTC) whether any of the submodules have a newer commit in the preferred branch. If so, several things will happen:

1. A new branch starting with *dependabot* will be created in the repository and any relevant workflows will be triggered.
2. A *pull request* will be created to pull the new branch into the *default* branch. This pull request must be manually reviewed and merged. Afterwards the *dependabot* branch can be deleted.

If the workflow `call-deploy-book` is used, and the *dependabot* branch should not be built and deployed (and all other branches you do want), you can achieve this by adding the next to the file `call-deploy-book.yml`:

```
on:
  push:
    branches:
    - '**'
    - '!dependabot**'
```

If you want another scheduled workflow, see Dependabot options reference for the options.

If you want to manually trigger the Dependabot workflow, you can do this by doing the next steps:

1. Go to your repository on Github.
2. Choose **Insights**.
3. Choose **Dependency graph**.
4. Choose **Dependabot**.
5. Choose **Recent update jobs** next to `.gitmodules`.
6. Choose **Check for updates**.

If you've private repositories, expand `.github/dependabot.yml` to:

```
version: 2
registries:
  submodule1-registry:
    type: "git"
    url: https://github.com/
    username: x-access-token
    password: ${{secrets.GH_PAT}}

updates:
  - package-ecosystem: "gitsubmodule" # See documentation for possible values
    directory: "/" # Location of package manifests
    schedule:
      interval: "weekly"
      day: "sunday"
      time: "23:59"
    registries:
      - submodule1-registry
```

`GH_PAT` should be a a personal access token added to the repository action secrets with 'repo' scope. If you're making use of the deploy-book-workflow, this is the same personal access token as required for cloning the submodules in the deploy-book-workflow

Build book on GitLab/GitHub with submodule

If you're using a GitLab/GitHub workflow, make sure you force it to fetch all the submodules as well. If you're using the TeachBooks GitHub/GitLab workflow, that has been taken care of.

Delete submodules

Deleting submodules is a bit notrocious... These steps https://www.baeldung.com/ops/git-submodule-add-remove proved to be useful:

1. Checkout to main

... using CLI | **... using GitHub Desktop**

```
git checkout main
```

2. Deinitialize submodule

```
git submodule deinit -f book/external/<external repository>
```

3. Remove submodule Git directory

Directly from file explorer or

```
rm -rf .git/modules/book/external/<external repository>
```

4. Remove from `.gitmodules`

Directly with text editor or:

```
git config -f .gitmodules --remove-section submodule.book/external/<external repository>
```

5. Stage Changes to `.gitmodules`

... using CLI | **... using GitHub Desktop**

```
git add .gitmodules
```

6. Remove from Git cache

```
git rm --cached book/external<external repository>
```

7. Commit and push changes

... using CLI | **... using GitHub Desktop**

```
git add .
git commit -m 'rm submodule: <external repository>'
git push
```

8. Delete repo and clone again For some reason the existence of the submodule is stored locally somewhere. To prevent further issues, you might want to delete the full parent repository and reclone it.

9. Merge with other branches Merge this change with all other branches which still have this submodule.

More info

Here you can find more information on Git submodules.

Contribute

This tool's repository is stored on GitHub. The `README.md` of the branch `manual-description` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

Auto-updating packages

Contents

When building your book, you are making use of various Python packages: the `teachbooks` and `jupyter-book` packages themselves, but also packages for extensions. These are regularly updated, however, those updates are not necessarily incorporated into your book automatically. The list of packages and their versions are defined in the `requirements.txt` file, which is provided as part of the template. Consider the following three options for how packages can be specified:

1. `requirements.txt` only contains names of packages like: `download_link_replacer`. In that case, your deploy-book-workflow will take the most up-to-date version when making your book website once a week (as the chache will be cleared once a week). This might lead to unexpected changes when a new version has been released (although new versions will generally be backwards compatible).
2. `requirements.txt` contains names of packages with a specified version like: `download_link_replacer==1.0.4`. In that case, your deploy-book-workflow always uses that specific version. In doing so, you'll never get a new update unless you explicitly adapt the version in `requirements.txt`. If you'd like to get notified for updates, you might consider using GitHub's Dependabot.
3. A combination of 1. and 2.: In that case (once a week at most) you will receive new versions for only the unfixed packages, no updates for the fixed versions.

For the case of specified versions, you can use GitHub's Dependabot to notify you that a new version is available *and* to automatically set up a Pull Request to update your book with the new version.

Notifications updated packages with Dependabot

Dependabot checks the specified version of packages in your `requirements.txt` file and, if a new version is found, will create a new branch, update the `requirements.txt` file and open a Pull Request whenever there's an update available for that package. Note that packages without a fixed version are ignored by Dependabot.

To activate this feature:

1. Specify version for all packages you want to be notified on in your `requirements.txt` file. See **`requirements.txt`** of this manual as an example
2. In the `.github/` directory, add a file named `dependabot.yml` with the following content (note that sphinx-thebe (used in python live coding) and docutils (using in APA referencing) are ignored because these require a very specific version to work):

```yaml
version: 2
updates:
  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "weekly"
      day: "sunday"
      time: "22:59"
    ignore:
      - dependency-name: "sphinx-thebe"
      - dependency-name: "docutils"
```

This check will run every Sunday around midnight (UTC) whether any of the fixed-version packages are updated. If so, several things will happen:

1. A new branch is created with a name that begins with `dependabot...` in the repository
2. A commit is made updating `requirements.txt` (e.g., `jupyterbook_patches==1.4.2` is changed to `jupyterbook_patches==1.4.4`)
3. A *pull request* will be created to merge the new branch into the *default* branch. This pull request must be manually reviewed and merged. Afterwards the *dependabot* branch can be deleted (automatically).

Note that these activities will occur automatically and may trigger other workflows in your repository (for example, the building of a book on another branch). If the workflow `call-deploy-book` is used, and you don't want the *dependabot* branches to be built and deployed (and all other branches you do want), you can achieve this by adding the following text to the file `call-deploy-book.yml`:

```
on:
  push:
    branches:
    - '**'
    - '!dependabot**'
```

If you want another scheduled workflow, see Dependabot options reference for the options.

If you want to manually trigger the Dependabot workflow, you can do this by doing the next steps:

1. Go to your repository on Github.
2. Choose **Insights**.
3. Choose **Dependency graph**.
4. Choose **Dependabot**.
5. Choose **Recent update jobs** next to  `requirements.txt`.
6. Choose **Check for updates**.

TeachBooks Versioning

Inspired by versioning systems in software management (i.e. semantic and CalVer versioning) TeachBooks suggests a versioning guideline in which the version number tells the reader about the impact of the change. This might more suitable for your online books than the more traditional way of versioning paper books (version 1, 2, etc.), because of the amount of edits which online books allow. We recommend TeachBooks versioning, which comes in two flavors:

1. `academic_year.additions.errata` versioning for books tailed-made for courses in which content is added / adapted during the course and might be restructured extensively every year while remaining to be available in the original form.
2. `major.errata` versioning for books which are more stable over years, in which big changes are covered only by the version number.

We have come up with guidelines how to use the two TeachBooks-versioning options (adapted from (Preston-Werner, 2025)):

academic_year.additions.errata | **major.errata**

1. A normal version number MUST take the form `academic_year.additions.errata` where `academic_year` is the academic year in which the book is used, and `additions`, and `errata` are non-negative integers, and MUST NOT contain leading zeroes. Each element MUST increase numerically. For instance: 2025.9.0 -> 2025.10.0 -> 2025.11.0
2. Once a versioned book has been released, the contents of that version MUST NOT be modified. Any modifications MUST be released as a new version.
3. Errata versions MUST be incremented if a small change is made which should be communicated to the reader in both the source code and in the book itself. If the small change is not crucial (like a simple typo), you might consider not defining it as a new version but combining it with other changes in the next errata or additions version.
4. Additions version MUST be incremented if an addition is made to the book. It MUST be communicated in both the source code and in the book itself. It MAY include errata level changes. The errata version MUST be reset to 0 when the addition version is incremented.
5. Academic year version MUST be incremented if a fresh book is started for which additions will follow later. It MUST be communicated in both the source code and in the book itself. Additions and errata versions MUST be reset to 0 when academic version is incremented.
6. Precedence refers to how versions are compared to each other when ordered.
    1. Precedence MUST be calculated by separating the version into academic year, additions and errata identifiers in that order.
    2. Precedence is determined by the first difference when comparing each of these identifiers from left to right as follows: Academic year, additions and errata versions are always compared numerically. Example: 2024.0.0 < 2025.0.0 < 2025.1.0 < 2025.1.1.
7. The way in which the version number is incremented after the initial release should be explained in the README of the source code and in the book itself.

An example can be seen in the source repository of the Engineering Systems Optimization book showing tags for different versions.

Read TeachBooks versioning with changelog on how to combine this with a changelog, tags and releases!

Additional functionality

This sections includes the features developed by TeachBooks which add functionality to the book. Most of them are implemented as sphinx extensions Sphinx Extension , which are plugins developed for the Sphinx documentation ecosystem (which Jupyter Book is built upon). They extend or modify Sphinx's capabilities. Since Sphinx extensions are not an integral part of jupyter-book, as opposed to original jupyter-book features, they must be installed and configured explicitly, for example in the `_config.yml` and `requirements.txt` files.

Download link replacer

Contents

You can control the download option of the book in two ways:

- Disabling downloading using cell tags `disable-download-page`
- Add / replace download link with custom link using sphinx-`download-link-replacer`

1. Disabling download

If you add `disable-download-page` as a cell tag to a cell in a python notebook, the download button (⬇) will disappear from the topright corner. The cell tag can be added to any code cell in the notebook. This function might be handy if your page includes code which you don't want the students to see. Be aware that this also removes the option to download a PDF of the page.

2. Add/replace download link

The download link ⬇ –> 📄 **.ipynb** can be replaced by using the following code in any markdown / notebook file:

```
```{custom_download_link} <link_target>
:text: "Custom text"
:replace_default: "True"
```
```

Replace `<link_target>` with the download location. It can either be a remote link (`http`, `https`, or `ftp`), or a local path (relative to the location of the file containing the directive). Local files must be located within or below the source folder of the book (i.e. the folder containing `_config.yml`).

The `replace_default` key is optional. When set to `True`, the default download link will be replaced with the custom one. When set to `False`, the default download link will be kept, and the custom one will be added below it. If the key is not set, the default behavior is to add the link to the list, without changing the default one.

The directive can appear multiple times in a single file.

A potential application of this functionality is when creating a page in which students have to do some coding. Downloading the page allows the student to save their work and work with their local environment. However, the original source file might include code (jupyterbook formatting, widgets, answers) which is not necessary for students. You can make a copy of the notebook file without these elements and replace the 📄 **.ipynb** link to this custom notebook file. Furthermore, you can add any additional data as an additional 📄 **.zip** file.

2.1. Installation

To install the Download-Link-Replacer follow these steps:

Step 1: Install the Package

Install the `download-link-replacer` package using `pip`:

```
pip install download-link-replacer
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
download-link-replacer
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions:

```
sphinx:
    extra_extensions:
        - download-link-replacer
```

Contribute

This tool's repository is stored on GitHub. The `README.md` of the branch `manual_docs` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

Multilingual book

Contents

The custom launch button extension allows you to create a customizable button in the top right of your jupyter book. This may have many applications, one of them being that you can create different language versions of the book available for the user.

What does it do?

This extension add a button to the top bar which allows you to link to other website. In combination with a translated book on another branch you can use this to create multilingual books.

Installation

1. Install the `sphinx-launch-buttons` package using `pip` if you'd like to build your book locally:

```
pip install sphinx-launch-buttons
```

2. Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
sphinx-launch-buttons
```

3. To use the extension in your book, add the extension to the list of Sphinx extra extensions in your `_config.yml` file, (**important**: underscore, not dash this time):

```
sphinx:
  extra_extensions:
    - sphinx-launch-buttons
```

Usage

This section will explain how to create a "Languages" button, like you might be used to seeing on just about any website. The use of the button, however, is completely customizable and may be used in many different ways.

1. Include a `_launch_buttons.yml` file in the same location (root directory of your book) as your `_config.yml` file. The following code cell shows the main structure of that file.

```
buttons:
  - type : dropdown

  - type : button
```

Here, `buttons` is an array of launch buttons, each can be identified using 2 types: 'dropdown' or 'button'. The cell above shows 2 buttons, one of type `dropdown` and one of type `button`.

The button/dropdown can be visualized using either an svg icon or text.

```
buttons:
  - type : dropdown
    label: Language
  - type : button
    icon : <svg> ...
           </svg>
```

2. Lastly you need to specify the items of your button. So assuming you want to have different language versions, each item will be one of the languages.

```
buttons:
  - type : dropdown
    label: Language

  - type : button
    icon : <svg> ...
           </svg>
    items:
      - label: English
        url: url of branch
      - label: Nederlands
        url: url of branch
```

As you can see in the `items:` line, each dropdown option links to the branch of the repository in the respective language.

Setting up your repository for multilingual book

For the implementation to your book, it is handy to create a branch for each language version you want to offer. Make a new branch using for example main as a source. Assuming we want to create a dutch version of you can call this branch `Dutch` or `Nederlands`.

You will then need to translate the content in the dutch branch to dutch which can take some time. From experience, DeepL is a good tool for this but any AI chatbot might be helpful as well. Make sure to proofread the translation.

You'll need to add (merge) the updated `_config.yml` and the new `_launch_buttons.yml` to all of your branches.

Example

```
buttons:
  - type: dropdown
    icon: <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-globe" viewBox
            <path d="M0 8a8 8 0 1 1 16 0A8 8 0 0 1 0 8m7.5-6.923c-.67.204-1.335.82-1.887 1.855A8 8 0 0 0 5.145 4H7.5zM4.
          </svg>
    items:
      - label: English
        url: https://teachbooks.github.io/files-and-folders/EN
      - label: Nederlands
        url: https://teachbooks.github.io/files-and-folders/NL
```

The code in the above cell is the `_launch_buttons.yml` file of a repository called “files-and-folders”. The buttons created look like this:

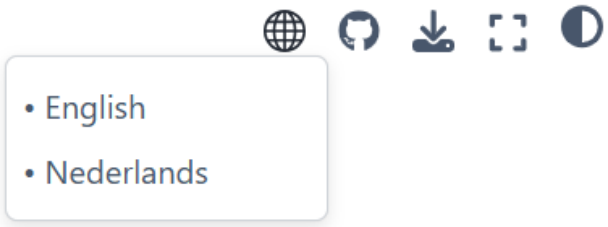


Fig. 107 Custom Button

An example of this usage in a book can be found in [this book called Files and Folders](https://teachbooks.io/files-and-folders/EN/intro.html}

Contribute

This tool’s repository is stored on GitHub. The `README.md` of the branch `manual` is also part of the TeachBooks manual as a submodule. If you’d like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

Discussions in your book: Utterances

Contents

Utterances 🔮 is a lightweight open-source widget which allows you, your colleages and your students to discuss stuff in a blog post in your book. It is build on GitHub issues, so requires a GitHub repository, although the book can be hosted anywhere (so also on GitLab).

The utterances website clearly explains the required steps. Three things to take care of are:

1. With the current setup of the deploy book workflow on GitHub, this widget only works on the primary branch.
2. The baseurl is the root url of your book (the part of the url that doesn't change when opening different pages of the book). It needs to be defined in the `_config.yml` so that utteranc.es knows where to redirect users while interacting with the widget:

```
html:
  baseurl :  "https://<user/organization>.github.io/<repo>" #Replace this with your own URL
```

3. It's advised to use `Issue title contains page pathname` as an option on utteranc.es, because that url is most stable.

The given script can be added anywhere in your book, just copy the html-script into your `.md`-file or a markdown cell in your `.ipynb`-file. The blogpost is not visible when you do a local build of the book, build it online or use a local python server as shown in Local Server to view interactive elements locally.

Users need to have a GitHub-account to post a message. Although this might be a burden, it allows students to track their previous posts and set up notifications on follow-up posts.

If you'd like notifications on new posts in your book, which end up as issues in your GitHub repository, you can configure the GitHub `watch` settings to do so as explained here.

Below you see an example of this feature!

Example discussion

H5p interactive elements

Contents

Instruction

H5p elements are interactive HTML-blocks which can be embedded in a Jupyter Book using an iframe. An H5p element can be created in the TU Delft portal on the H5p website (sign in via Brightspace to H5p required first). The iframe code can be copied at Edit - Publish - Public - Embed code. This html code can be directly added to your markdown file using Iframes:

````
```{h5p} https://....h5p.com/content/.../embed
```
````

Alternative, you can use raw html:

```
<iframe src="https://....h5p.com/content/.../embed" aria-label="..." width="1088" height="200" frameborder="0" allowfull
```

A full list of all available interactive elements is available on the website of H5p, these include among others:

- Multiple choice questions
- Fill in the blanks
- Drag and drop
- Interactive video
- Image hotspot

Some best-practices on the use of H5p:

- H5p requires its own hosting. TU Delft provides this for TU Delft employees.
- To startup your H5p-account, first login to H5p via Brightspace. You can follow instructions as under https://www.tudelft.nl/teaching-support/educational-tools/h5p.
- After that, direct login is possible via tudelft.h5p.com
- Place your H5p-elements in a shared folder in H5p.
- Disable the display options “Toolbar Below Content” and “Display author’s name to public (anonymous users) (only relevant when content’s status is set to Public)” for each element for a smooth experience. On the other hand, the options ‘Toolbar below content’ in combination with ‘Allow users to download content’ allow people to reuse your questions, making them more open.
- Use it in combination with Iframes to allow for a proper dark mode, dynamic scaling and a transparent background.

In the next subpage, these exampled are shown.

More information on H5p using the TU Delft provided hosting: Teaching Support - Educational Tools - H5P

Examples of H5p Quizzes

Contents

Dialog cards

Dialog cards can be used to ask open questions about the content students are reading or, more intrestingly, to check the conclusion the students draw from the interactive exercises built into the book.

Engineering Systems Optimization

A great example of this can be found in the book: Engineering Systems Optimization. Chapter 4 called Mulit-objective optimization has an in depth sub-chapter which is used to compare the relation between engine power and emissions and how to optimize for both.

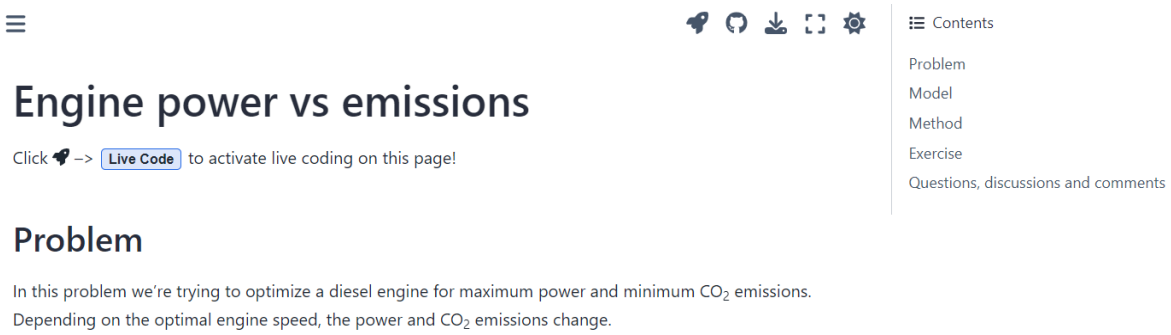


Fig. 108 Problem Statement

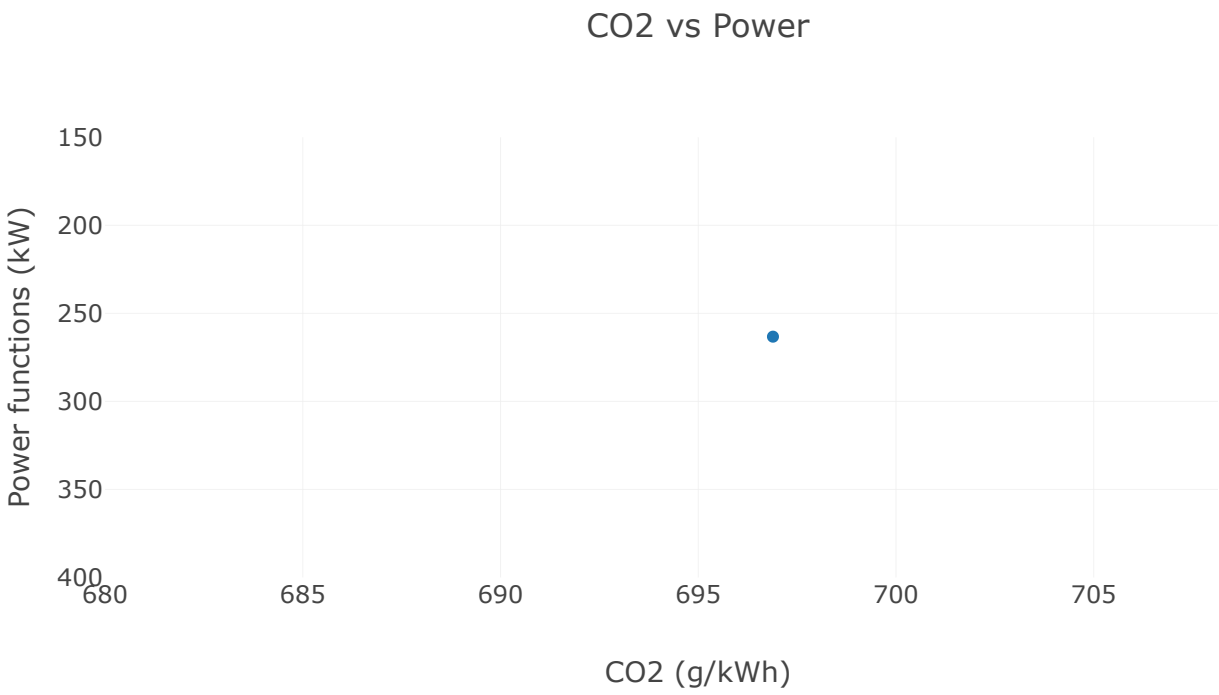
As can be seen on the right-hand-side menu, the sub-chapter includes the problem statement, the elaboration on the model and solution method and finally an exercise for the student. The code in the page is activated by pressing the live code button which will make

the code cells editable and executable. Additionally, interactive figures are included into the text with movable sliders to directly see the effect of the changes made.

RPM 1000

Weighted Objective: = - 1/3 * POWfunc(1000) + 2/3 * CO2func(1000) = - 1/3 * 263.24 + 2/3 * 696.89 = 376.85

Goal Attainment: max(460 - POWfunc(1000), CO2func(1000) - 640) = max(460 - 263.24, 696.89 - 640) = 196.76



The H5p quizzes can shine in the section on questions, discussions and comments. Next to many multiple choice questions (which are discussed later) there are dialog cards which can ask more conceptual and open ended questions. Pressing on the blue button `Turn` will flip around the card and reveal the model answer. The great thing about the integration of both interactive aspects (coding and questions) helps the student to play around with the code and understand the topic from a less theoretical point-of-view.

Question Set

Question sets consist of a question and one or more possible answers to that question.

MUDE - Observation Theory

The following question sets are taken from chapter 3.3. Weighted least-squares estimation of the MUDE 23/24 book. The chapter builds on the previous chapter on simple least-squares estimation and dwelves into the importance and properties of the weight matrix. The

student is primarily quizzed using multiple choice questions to check their understanding. The answers can be provided in the shape of text, figures, equations, matrices and probably much more.

Select the correct statement(s).

- [ ] The weighted least-squares solution provides the smallest possible \ (\hat{\epsilon}^T\hat{\epsilon}\).
- [ ] If we apply weighted least-squares, we 'expect' the residuals of observations with smaller weight to be larger.
- [ ] Ordinary least-squares estimation is also a form of weighted least-squares.

Check

You and your fellow students are asked to measure the width of a canal. You all decide to take a different approach. Alice is asked to come up with an estimate of the canal width using all \(m\) observations. Based on her own 'ranking', Alice decides to give different weights to the different observations. Which of the following is the equation that Alice will use to estimate the canal width? The individual weights are called \(w_i\), \((i=1,..., m) \)

- ○ \(\hat{x} = \sum_{i=1}^{m} \frac{y_i}{w_i} \)
- ○ \(\hat{x} = \frac{1}{\sum_{i=1}^{m}w_i}\sum_{i=1}^{m}y_i \)
- ○ \(\hat{x} = \frac{1}{\sum_{i=1}^{m}w_i}\sum_{i=1}^{m}(w_i y_i) \)
- ○ \(\hat{x} = \frac{1}{m}\sum_{i=1}^{m}y_i \)

Check

Since these questions are meant to develop the understanding of the students, feedback is vital! Feedback can be built into the questions to explain the reasoning behind a correct answer.

$$\hat{x} = \sum_{i=1}^{m} \frac{y_i}{w_i}$$

✔ $$\hat{x} = \frac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} (w_i y_i)$$

$$\hat{x} = ([1 \quad \dots \quad 1] \begin{bmatrix} w_1 & & \\ & \ddots & \\ & & w_m \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix})^{-1} [1 \quad \dots \quad 1] \begin{bmatrix} w_1 & & \\ & \ddots & \\ & & w_m \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Note: this is the equation for the weighted average! Hence, the weighted least-squares is equal to the weighted average in case we have observations with $\mathbb{E}(Y_i) = x$ ($\forall i = 1, \dots, m$).
Check yourself what happens if all weights are identical.

$$\hat{x} = \frac{1}{m} \sum_{i=1}^{m} y_i$$

$$\hat{x} = \frac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} y_i$$

Keep going!

1/1

Fig. 109 Correct Answer Feedback

And when the answer is not correct, tips can be given to nudge the student towards the right solution.

Which of the following expressions is correct?

✖ $\hat{y}_i = 0.95 + 2.5t_i$

Did you use $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}$ and $\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$?

$\hat{y}_i = 0.97 + 2.5t_i$

$\hat{y}_i = 1.0 + 2.4t_i$

0/1 Show solution Retry

Fig. 110 Tips

Try it out!

Have a look at the following 7 questions.

Complex fill in the blanks

The complex fill in the blanks can be used to evaluate numerical answers.

MUDE - Observation Theory

Chapter 3.7. Non-linear least-squares estimation from the MUDE 23/24 book makes use of the complex fill in the blanks quiz in order to check that the students understood the composition of the Jacobian Matrix specifically used in deriving the linearized functional model. This model is essential for accurately estimating values and assessing system behavior of complex processes that arent linear.

As you can see, the precision of the answer sought for is specified and the figure below shows again the incorporation of feedback.

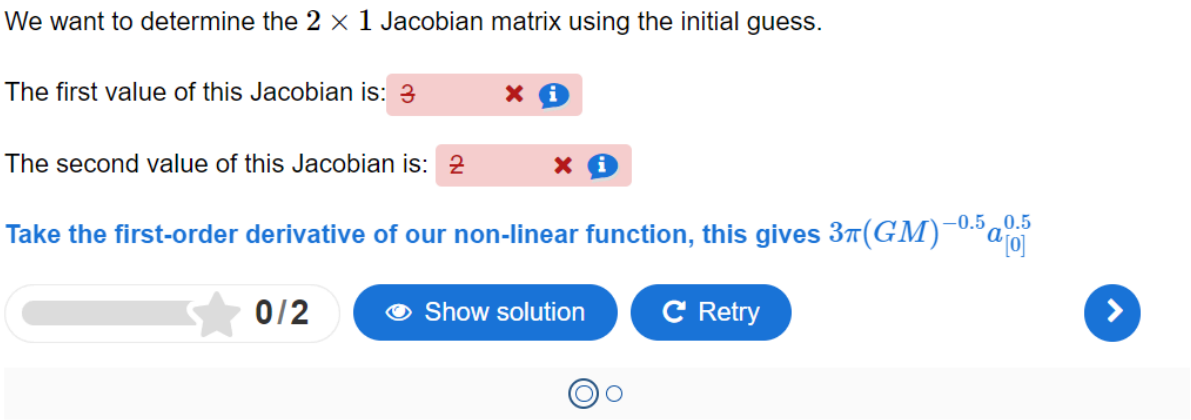


Fig. 111 Wrong Answer + Feedback

MUDE - Optimization

Since the cells can be used to evaluate numerical answers, they are an easy way to quiz the student on matrices and evaluate the result of each position independently. The optimization team of MUDE took this a step further and used this set-up to quiz the students on the SIMPLEX method. This method automates the solving of the augmented form of a linear programming problem with continuous variables. It makes use of tables and

the manipulation of equations to find the optimal values. Unfortunately, with a large number of variables this tables tend to get quite big and filling in the cells becomes laborious.

Build the first table of the simplex method.

	\(Z\)	\(x_1\)	\(x_2\)	\(s_1\)	\(s_2\)	\(s_3\)	\(b\)
\(Z\)							
\(S_1\)							
\(S_2\)							
\(S_3\)							

Truss structure

Truss structures are modelled as rigid bars (so elements which cannot deform) connected by hinges (so elements can rotate with respect to one each other). In our model, hinges are indicate with a circle, and bars with a line. For example, the structure you've seen in the second example (with two diagonal bars removed) is modelled as follows:

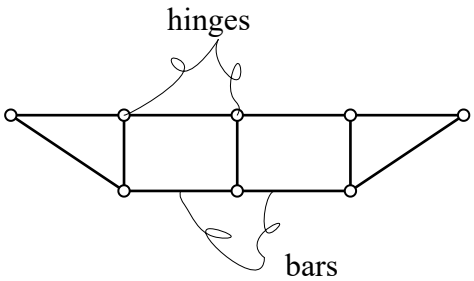


Fig. 112 Bars and hinges in truss structure

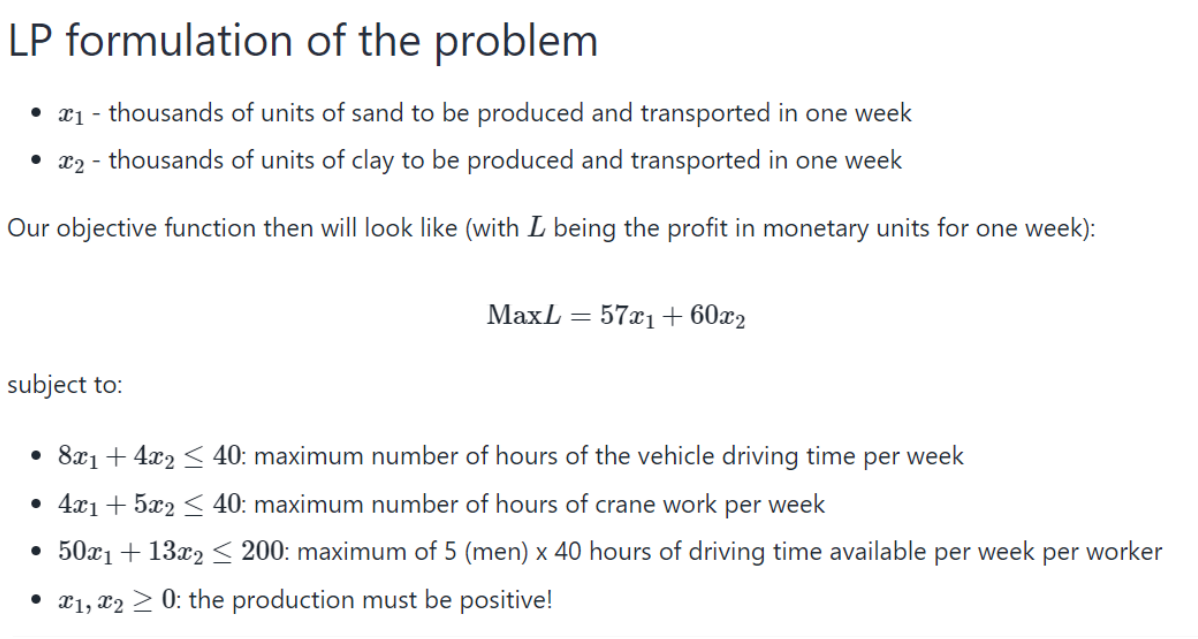
Now that you've been introduced to truss structures, answer the following question:

True or False

This next type of quiz is fairly self explainatory :)

MUDE - Optimization

Once again the chapter on optimization in the MUDE book integrated many true and false questions to test the students on the problem statement which, like in many engineering problems, is just as important if not more important than the solution procedure. This is especially when the case when small tweaks to the equations, such as seen in the figure below, can change the result of the optimization drastically.



LP formulation of the problem

- x_1 - thousands of units of sand to be produced and transported in one week
- x_2 - thousands of units of clay to be produced and transported in one week

Our objective function then will look like (with L being the profit in monetary units for one week):

$$\text{Max} L = 57x_1 + 60x_2$$

subject to:

- $8x_1 + 4x_2 \leq 40$: maximum number of hours of the vehicle driving time per week
- $4x_1 + 5x_2 \leq 40$: maximum number of hours of crane work per week
- $50x_1 + 13x_2 \leq 200$: maximum of 5 (men) x 40 hours of driving time available per week per worker
- $x_1, x_2 \geq 0$: the production must be positive!

Fig. 113 Problem Statement

The small circles below the question indicates that there are more questions in this series which can be viewed by clicking on the blue arrow pointing towards the left or right. Open the second, fourth and fifth question:

Try it out!

Let's consider an estimation problem with m observables and functional model ($\mathbb{E}(Y)=\mathrm{Ax}$) and stochastic model $\Sigma_Y = \sigma^2 I_m$ (i.e., the covariance matrix is a scaled identity matrix).

Find the expressions for the ordinary (unweighted) least squares estimator and the best linear unbiased estimator.

True or False:

The ordinary least squares and best linear unbiased estimator are equal to each other.

○ True ○ False

Check

Drag and Drop

Drag and Drop questions are useful to make the distinction between different concepts that might often cause confusion.

MUDE - Optimization

In this chapter, it was important that the students made the distinction between the terminology and mathematical meaning between different elements of the graphical solution method. A drag and drop question was used to link the term to its definition. The answer boxes can be dragged to the correct title in a playful manner.

The fourth question in this question set is a drag and drop question:

The intersection of the gradient vector with the feasible solution space polygon determines the optimal solution.

○ True ○ False

Check

The correct answer leads to:

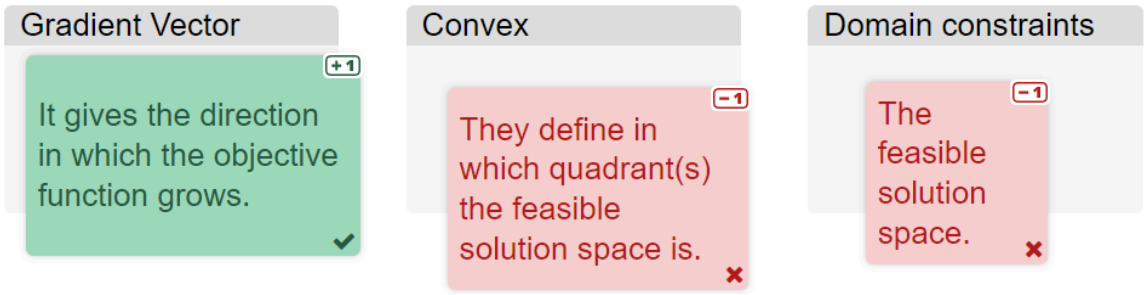


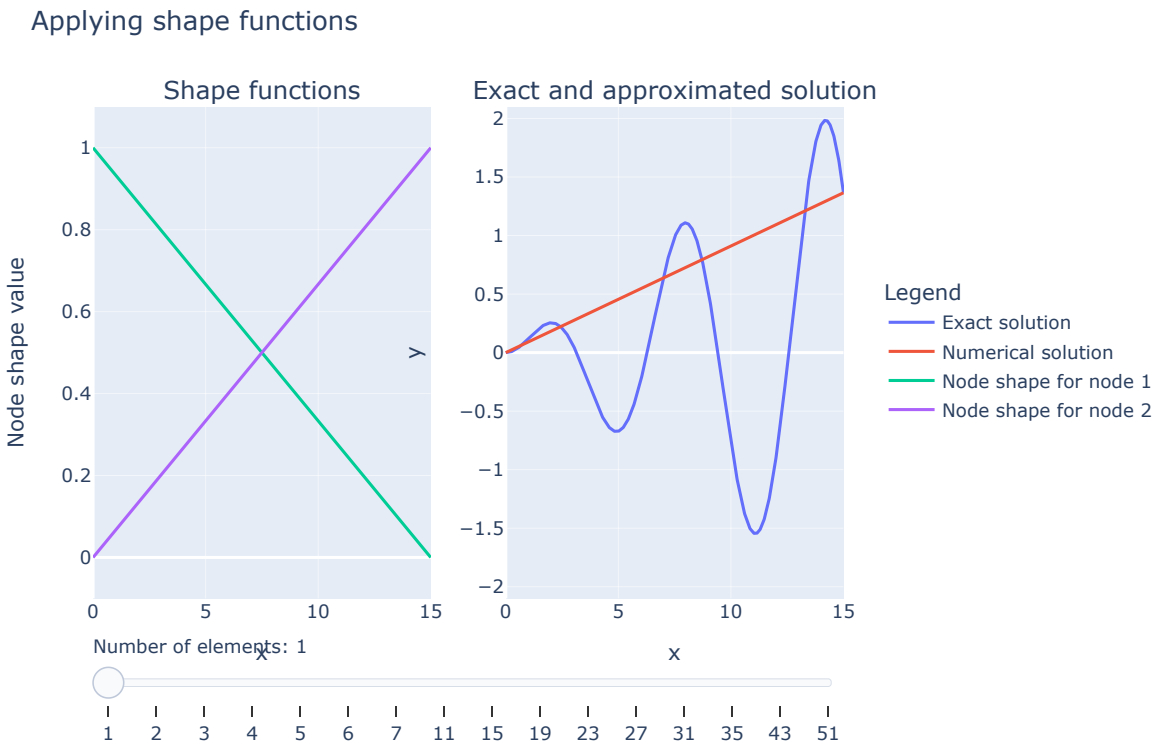
Fig. 114 Solution

Interactive plots: plotly

The use of the plotly package is an example which allows you to create HTML/Javascript widgets from python code without any knowledge on HTML/Javascript.

The example below is taken from the FEM-module by Frans van der Meer. When using Plotly graphs, make sure the notebook is executed in Jupyter Lab / Jupyter Notebook for the figures to be shown in the book. Running the code in VS code might break the result in the book, although the output is visible in VS code. Alternatively, add the following lines of code to your notebook to have a valid output in the book as well:

```
import plotly.io as pio
pio.renderers.default = 'notebook'
```



Grasple

Contents

This package contains a Sphinx extension for inserting Grasple exercises into a Jupyter book as an iframe. It allows you to easily add Grasple question with some formatting and, more importantly, the creation of QR codes in the PDF version of the page. This leads to the source link of the iframe.

This package is a continuation of the package dbalague/sphinx-grasple.

Grasple gives you an embed code for each exercise, which can be added directly to your markdown-file. This package improved the embedding. More information on Grasple and the support of TU Delft can be found on: Teaching Support - Educational Tools - Grasple

Installation

To install the teachbooks-sphinx-grasple extension, follow these steps:

Step 1: Install the Package

Install the `teachbooks-sphinx-grasple` package using `pip`:

```
pip install teachbooks-sphinx-grasple
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
teachbooks-sphinx-grasple
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions (**important**: underscore, not dash this time):

```
sphinx:
    extra_extensions:
        - teachbooks_sphinx_grasple
```

Usage

To use, include the following in your Jupyter book

```
::::{grasple}
:iframeclass: dark-light
:url: https://embed.grasple.com/exercises/f6c1bb4b-e63e-492e-910a-5a8c433de281?id=75093
:label: grasple_exercise_1_3_4
:dropdown:
:description: Cross product in $R^4$?

::::
```

In the jupyter book this leads to:

Grasple Exercise 1

Cross product in R^4?

In the PDF this leads to:

Grasple Exercise 1

Example of exercise which goes along

Note that the iframe doesn't load locally in VSCode or in the build file, only in the online book or if shown in a local server.

Important Note

The tests provided are still the original ones from sphinx-exercise and have not (yet) been adapted.

Contribute

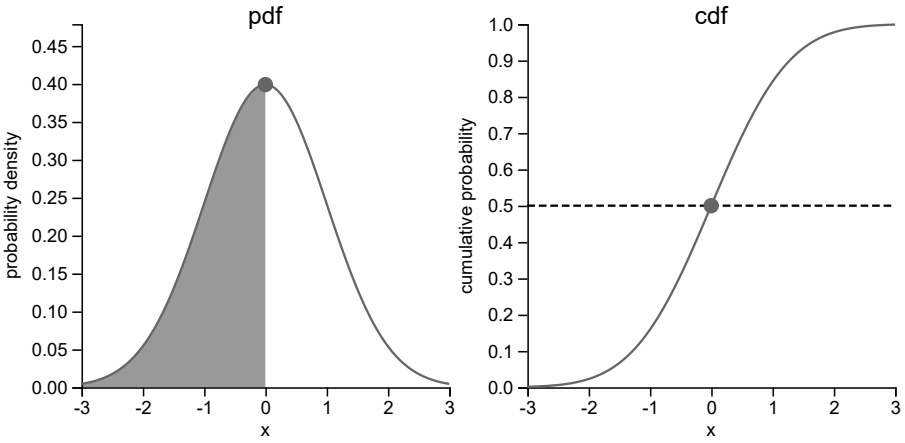
This tool's repository is stored on GitHub. The `README.md` of the branch `manual_docs` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

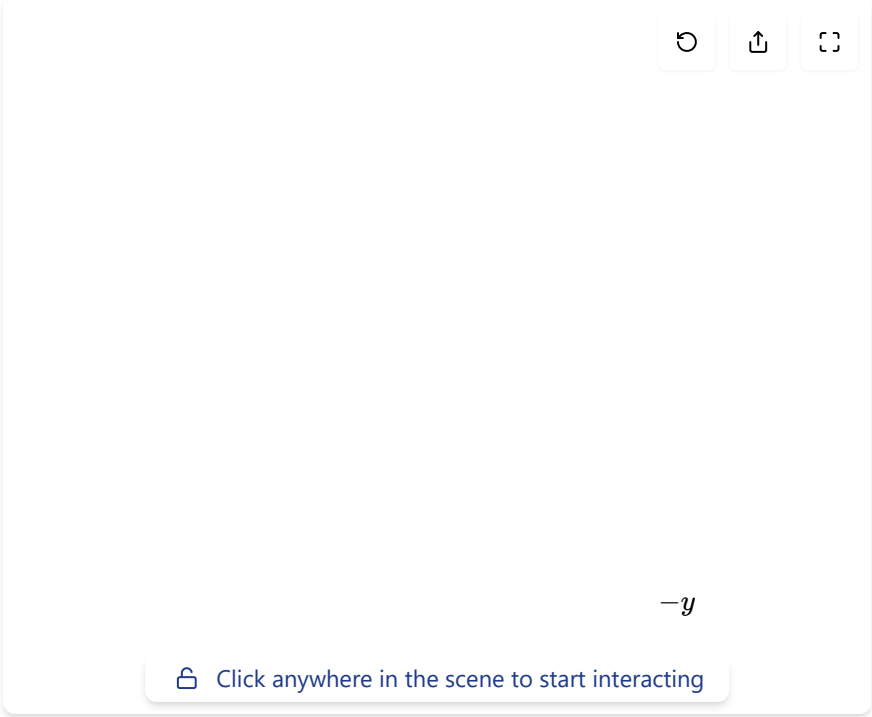
Interactive HTML/JavaScript elements

In this section, we will discuss how to create and embed interactive HTML/Javascript elements created in HTML/JavaScript in a TeachBook. These HTML/Javascript element work very fluently and don't require a python kernel running in the background.

This could be for example interactive 3d figures:

Or other interactive graphs:




$-y$
Click anywhere in the scene to start interacting

Adding interactive HTML/JavaScript elements

Contents

In this section, we will discuss how to embed interactive elements created in HTML/JavaScript in a TeachBook. If you are interested in learning how to create these elements yourself, we will discuss an example in the next article. As an example, let us discuss how to embed an interactive 3D model of a subsurface environment I created:

Storing the HTML file

TeachBooks like this one are built from a Git project, comprising of different files and folders that are assembled into a webpage. It is generally not possible to paste the code of your HTML element directly into one of the project's Markdown files. Instead, you will have to store the files separately somewhere in the folder structure, then embed them later on. For this project, the book's root folder contains a ´_static´ folder which we can use to store our HTML/JS file. Some elemenets - such as the one above - may require additional files. The HTML element above, for instance, specifies a renderer using THREE.js that displays a digital model. This model is specified as an .obj file (which defines its geometry) and a .mtl file (which defines its materials - its colors, reflectivity, transparency, and so on). We can store these files in the same ´_static´ folder.

Embedding HTML files via iframes

If we want to embed our HTML/JS element in a Markdown page, we must use an inline frame, also known as an iframe. In a book, an iframe can be added using Iframes:

```
```{iframe} ../_static/element_render_box.html
:width: 600px
:height: 300px
```
```

or alternatively using raw HTML:

```
<iframe src="../_static/element_render box.html" width="600" height="300" frameborder="0"></iframe>
```

What does this code do? In HTML, certain active parts of the code are defined as tags surrounded by `<` and `>` symbols. The tag `<iframe>` opens the iframe environment, the `<iframe/>` tag closes it again. Inside the opening tag, we can further specify properties of the iframe. These properties are usually defined as strings, that is to say, surrounded by quotation marks. Some of the more common iframe properties are:

- `src`: This specifies the **source** of the iframe, the HTML file it will display.
- `width`: This specifies the **width** of the iframe in pixels.
- `height`: This specifies the **height** of the iframe in pixels.
- `frameborder`: This flag specifies whether a border should be drawn around the iframe (`"1"`) or not (`"0"`).
- `scrolling`: Specifies whether or not to display scrollbars for the iframe, in case the embedded content is larger than the iframe. Possible values are `"yes"`, `"no"`, and `"auto"`. I recommend either creating HTML elements of the same size, or adjusting their size dynamically to their contained - in this case, the iframe.
- `allowfullscreen`: Specifies whether or not to allow the iframe to be displayed in fullscreen mode.
- `loading`: Specifies how the browser should handle the loading of the iframe content. Possible values are eager (load immediately) and lazy (load on demand).

Let's take a closer look at the source. Since this project uses a local file structure, we can make use of relative paths such as `../_static/element_render box.html`. What does this path do? In this project, the markdown file for this article is located in the directory `book/features/adding_HTML_elements.md`. The HTML element we want to embed is located in the directory `book/_static/element_render_box.html`. The relative path thus works as follows:

- We start in the markdown file's directory `book/features/adding_HTML_elements.md`
- The string `"../"` tells the iframe to move up one folder to `book`. If you need to move up more subdirectories, use it recursively, eg. `../../` moves up two directories.
- The following string `"_static/"` tells the iframe to navigate into the `_static` folder. We are now in `book/_static/`.
- Finally, the string `"element_render_box.html"` selects the HTML file we want to embed. We are now in `book/_static/element_render_box.html`.

Viewing result at local build

To view the final result locally, it's required to setup a local Python webserver.

HTML code for the 3D render element

To conclude, I provide the HTML/JavaScript code for the 3D render element above. To understand what the code below does in more detail, please refer to the next article. One thing I want to direct your attention to are the paths specified in the first few lines of the `<script>` environment. Here, we do not have to navigate to a different local folder because we initially stored our `.obj` and `.mtl` files in the same folder as the `element_render_box.html` file. If they would be stored in a different folder, we have to specify either an absolute path (e.g., via a URL) or a relative path, as explained above.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Subsurface</title>
  <style>
    html, body {
      margin: 0;
      height: 100%;
    }
    #c {
      width: 100%;
      height: 100%;
      display: block;
    }
  </style>
</head>
<body>
  <canvas id="c"></canvas>

  <script type="module">

        // Path to object
    var path_obj = "bimodal_field.obj";
    var path_mtl = "bimodal_field.mtl";


    // Three.js - Load .OBJ and .MTL file - Windmill2
    // from https://threejsfundamentals.org/threejs/threejs-load-obj-materials-windmill2.html

    import * as THREE from 'https://threejsfundamentals.org/threejs/resources/threejs/r127/build/three.module.js';
    import { OrbitControls } from 'https://threejsfundamentals.org/threejs/resources/threejs/r127/examples/jsm/controls/
    import { OBJLoader } from 'https://threejsfundamentals.org/threejs/resources/threejs/r127/examples/jsm/loaders/OBJLo
    import { MTLLoader } from 'https://threejsfundamentals.org/threejs/resources/threejs/r127/examples/jsm/loaders/MTLLo

    function main() {
      const canvas = document.querySelector('#c');
      const renderer = new THREE.WebGLRenderer({ canvas });

      const fov = 45;
      const aspect = 2; // the canvas default
      const near = 0.1;
      const far = 100;
      const camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
      camera.position.set(0, 10, 20);

      const controls = new OrbitControls(camera, canvas);
      controls.target.set(0, 5, 0);
      controls.update();

      const scene = new THREE.Scene();
      scene.background = new THREE.Color('white');

      {
        const skyColor = 0xB1E1FF; // light blue
        const groundColor = 0xB97A20; // brownish orange
        const intensity = 1;
        const light = new THREE.HemisphereLight(skyColor, groundColor, intensity);
        scene.add(light);
      }

      {
        const color = 0xFFFFFF;
        const intensity = 1;
        const light = new THREE.DirectionalLight(color, intensity);
        light.position.set(5, 10, 2);
        scene.add(light);
        scene.add(light.target);
      }

      function frameArea(sizeToFitOnScreen, boxSize, boxCenter, camera) {
        const halfSizeToFitOnScreen = sizeToFitOnScreen * 0.5;
        const halfFovY = THREE.MathUtils.degToRad(camera.fov * .5);
        const distance = halfSizeToFitOnScreen / Math.tan(halfFovY);
        // compute a unit vector that points in the direction the camera is now
        // in the xz plane from the center of the box
        const direction = (new THREE.Vector3())
          .subVectors(camera.position, boxCenter)
          .multiply(new THREE.Vector3(1, 0, 1))
          .normalize();

        // move the camera to a position distance units way from the center
```

```
        // in whatever direction the camera was from the center already
        camera.position.copy(direction.multiplyScalar(distance).add(boxCenter));

        // pick some near and far values for the frustum that
        // will contain the box.
        camera.near = boxSize / 100;
        camera.far = boxSize * 100;

        camera.updateProjectionMatrix();

        // point the camera to look at the center of the box
        camera.lookAt(boxCenter.x, boxCenter.y, boxCenter.z);
      }

      {
        const mtlLoader = new MTLLoader();
        mtlLoader.load(path_mtl, (mtl) => {
          mtl.preload();
          const objLoader = new OBJLoader();
          objLoader.setMaterials(mtl);
          objLoader.load(path_obj, (root) => {
            scene.add(root);

            // compute the box that contains all the stuff
            // from root and below
            const box = new THREE.Box3().setFromObject(root);

            const boxSize = box.getSize(new THREE.Vector3()).length();
            const boxCenter = box.getCenter(new THREE.Vector3());

            // set the camera to frame the box
            frameArea(boxSize * 1.2, boxSize, boxCenter, camera);

            // update the Trackball controls to handle the new size
            controls.maxDistance = boxSize * 10;
            controls.target.copy(boxCenter);
            controls.update();
          });
        });
      }

      function resizeRendererToDisplaySize(renderer) {
        const canvas = renderer.domElement;
        const width = canvas.clientWidth;
        const height = canvas.clientHeight;
        const needResize = canvas.width !== width || canvas.height !== height;
        if (needResize) {
          renderer.setSize(width, height, false);
        }
        return needResize;
      }

      function render() {

        if (resizeRendererToDisplaySize(renderer)) {
          const canvas = renderer.domElement;
          camera.aspect = canvas.clientWidth / canvas.clientHeight;
          camera.updateProjectionMatrix();
        }

        renderer.render(scene, camera);

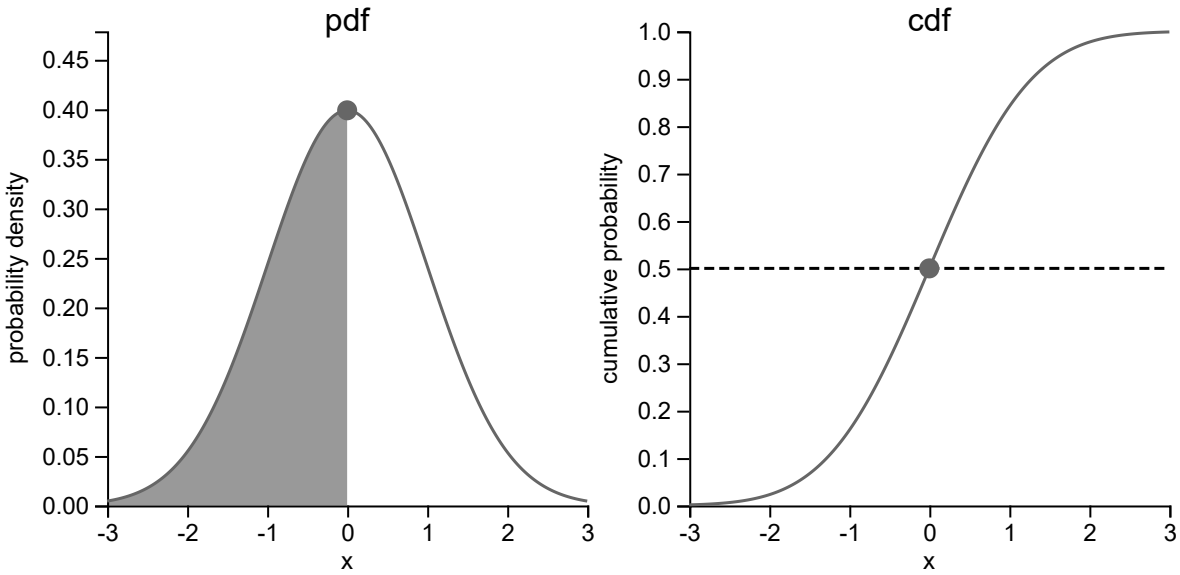
        requestAnimationFrame(render);
      }

      requestAnimationFrame(render);
    }

    main();
  </script>
</body>
</html>
```

Creating basic interactive HTML/JavaScript elements

Contents



At the bottom of this page, I will provide the full code for the interactive element above. In this section, let us go through the code one piece at a time. Let us begin from the top.

In the first line, we specify that this file is a HTML file. In the header (`<head>`), we can load packages we need later on. Within the `<script>` tags, we are loading three different libraries from web servers: `math.js`, which provides functions for basic mathematical operations (similar to Python's `numpy` library), as well as two versions of the library `d3.js`, which contains many extremely useful functions for web interactivity.

The `<style>` tag below defines CSS properties that, unsurprisingly, style the element. Two common settings for the `body` are `margin: 0`, which makes sure that the useable space extends all the way to the edges, and `overflow: hidden`, which effectively clips content that extends outside the element's borders by removing the scrollbars.

```
<!DOCTYPE html>
<html>
   <head>
      <script src="https://unpkg.com/mathjs/lib/browser/math.js"></script>
      <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.17/d3.min.js"></script>
      <script src="https://d3js.org/d3.v4.min.js"></script>
      <style>
         body{
         margin: 0; overflow: hidden;
         font-family: Helvetica, sans-serif;
         }
      </style>
   </head>
```

Next, we create a `<div>`. This is an HTML element that groups other HTML elements. In our case, it only contains a SVG named `"click"`. This SVG will serve as the canvas on which we will later place the interactive content.

```
   <!-- Create a div where the graph will take place -->
   <div id="my_datavisualization">
      <svg id="click" xmlns="http://www.w3.org/2000/svg">
      </svg>
   </div>
```

So far, all of this has just been setup. The real magic begins in the HTML's `<body>`, which contains the `<script>` environment. This is where we code the actual content of the interactive element in JavaScript.

The first few lines of our JavaScript code are simple. To scale our element with the size of its container (the iframe we created above), the code first reads the width `vw` and height `vh` of its container. For this element, we define the interactive element to be twice as wide as tall, so we define the `height` of the canvas as half its `width`. We then select the `"click"` canvas we created before, create a variable for quick access called `svg`, and set its height and width to the new dimensions we just computed.

```
<body>
   <script>
      // Get the viewport height and width
      const vw = Math.max(document.documentElement.clientWidth || 0, window.innerWidth
      const vh = Math.max(document.documentElement.clientHeight || 0, window.innerHeig

      // Fit to viewport
      var height = vw * 0.5;
      var width = vw;

      // Create the canvas. We will use only part of it for the main plot
      var svg = d3.select("#click") // This selects the div
          .attr("width", width) // This defines the canvas' width
          .attr("height", height) // This defines the canvas' height
```

Next, we may want to define some functions for our element. The first function `linspace` is simply creates `resolution` points between a `start` and `end` value, similar to `numpy.linspace` in Python. The second and third functions evaluate the pdf and cdf of a standard Gaussian distribution for a given `x` value.

```
function linspace(start, end, resolution) {
    var spacing = [];
    // Go through a for-loop
    var i;
    for (i = 0; i < resolution; i++) {
        spacing.push(start + (end - start) * i / (resolution - 1))
    }
    return spacing; // The function returns the linspace between p1 and p2
}

function standard_Gaussian_pdf(x) {
    mean = 0;
    std = 1;
    // Create a dummy variable
    var result = [];
    normalize = math.dotDivide(
        1,
        math.sqrt(
            math.dotMultiply(
                math.dotMultiply(
                    2,
                    math.PI),
                math.pow(
                    std,
                    2))));

    var i;
    for (i = 0; i < x.length; i++) {
        // Evaluate the first element of the Gaussian mixture
        expon = math.exp(
            -math.dotDivide(
                math.pow(
                    x[i] - mean,
                    2),
                math.dotMultiply(
                    2,
                    math.pow(
                        std,
                        2))))
        temp = math.dotMultiply(
            normalize,
            expon)
        result.push(temp)
    }
    return result
}

function standard_Gaussian_cdf(x) {
    mean = 0;
    std = 1;
    // Create a dummy variable
    var result = [];
    for (i = 0; i < x.length; i++) {
        // Evaluate the first element of the Gaussian mixture
        temp = 0.5 * (1 + math.erf(x[i] /math.sqrt(2)))
        result.push(temp)
    }
    return result
}
```

Now, let us prepare the two subplots for the pdf (left side) and cdf (right side). First, we define what range of values we want to plot on the x-axis and y-axis. On the x-axis, we want to plot values between -3 and +3. For the pdf subplot's yaxis, we want to see everything from `0` to the pdf*s maximum at `x=0`, plus 20% padding. In JavaScript, we have significant control on how we want to create our plot. This means that we must define where exactly we want to place our subplots on the canvas. Let us start by defining the horizontal width of the left subplot from 10% of the canvas's width to 50% of the canvas's width, and its height from 10% of the canvas's height to 89% of the canvas's height (recall that our canvas is twice as wide as it is tall). Mind that in Javascript, the point at 0% width and 0% height is located in the top-left corner.

Next, we create three helper functions. If we want to add an element to the canvas, we must define its position and size in pixel values. For plotting, this is inconvenient, so here we define three functions that

- convert x-values to horizontal pixels (`xScale_pdf`),
- convert horizontal pixels to x-values (`xScale_pdf_inverse`), and
- convert pdf-values to vertical pixels (`yScale_pdf`).

```
// Define a subplot for the standard normal
const x_limits = [-3, 3];
const y_limits_pdf = [0, 1 / math.sqrt(2 * math.pi) * 1.2];

const window_x_pdf = [width * 0.1, width * 0.5];
const window_y_pdf = [height * 0.1, height * 0.89];

// Get scaling functions for the x scale and the y_scale
const xScale_pdf = d3.scaleLinear()
    .domain([x_limits[0], x_limits[1]])
    .range(window_x_pdf)
const xScale_pdf_inverse = d3.scaleLinear()
    .domain(window_x_pdf)
    .range([x_limits[0], x_limits[1]])
const yScale_pdf = d3.scaleLinear()
    .domain([y_limits_pdf[0], y_limits_pdf[1]])
    .range([window_y_pdf[1], window_y_pdf[0]])
```

With these functions in hand, let us draw the first subplot. We start by drawing the x-axis and its label first, where we define the font-sizes relative to the canvas's width to ensure the font scales with its container's size. Next, we do the same for the y-axis. Finally, we add a label on top.

```
        // Draw the x axis
        svg
            .append("g")
            .attr("transform", "translate(0," + window_y_pdf[1].toString() + ")")
            .call(d3.axisBottom(xScale_pdf).ticks(5))
            .style("font-size", (12 * width / 600).toString() + "px")
        svg.append("text")
            .attr("transform",
                "translate(" + (math.mean(window_x_pdf)).toString() + "," + (height * 0.
            .style("text-anchor", "middle")
            .text("x")
            .style("font-size", (12 * width / 600).toString() + "px")

        // Draw the y axis
        svg
            .append("g")
            .attr("transform", "translate(" + (window_x_pdf[0]).toString() + ",0)")
            .attr("id", "mainxaxis")
            .call(d3.axisLeft(yScale_pdf))
            .style("font-size", (12 * width / 600).toString() + "px");
        svg.append("text")
            .attr("transform",
                "translate(" + (width * 0.03).toString() + "," + (math.mean(window_y_pdf
            .style("text-anchor", "middle")
            .text("probability density")
            .style("font-size", (12 * width / 600).toString() + "px")

        // Draw the subplot label
        svg.append("text")
            .attr("transform",
                "translate(" + ((window_x_pdf[1] - window_x_pdf[0])/2 + window_x_pdf[0])
            .style("text-anchor", "middle")
            .text("pdf")
            .style("font-size", (16 * width / 600).toString() + "px")
```

Now, let us add actual content. We start by creating 201 equally-spaced points on the x-axis between -3 and +3 using the `linspace` function we created above, then evaluate the corresponding densities using our `standard_Gaussian_pdf`. The first real feature we add to the subplot is a simple round marker that indicates where we currently are on the pdf. We can add geometric shapes via a syntax like `svg.append("circle)`, then define its attributes via `.attr()`. Defining a circle requires a radius attribute `r`, a horizontal position `cx`, and a vertical position `cy`. We initially place our marker at `x=0`. Note the use of the Scale functions we created earlier to convert subplot coordinates into canvas pixel coordinates. The `fill` and `stroke-width` properties define the style of the marker. An important property is the `id`, which we can use later to select and update this marker.

Next, we concatenate our `x` and `pdf` vectors into a list of dictionaries, which we then convert into pixel coordinates with a `valueline_pdf` function we create. Finally, we add it to the subplot as a `path` via the `svg.append("path")` function. In addition, we want to represent the fact that the cdf (which we will later plot on the right) represents the integral of all pdf values to the left of the current x-value. To this end, we want to create a filled shape that

highlights this area. We can do so by creating another path that follows the pdf from the left border to the currently selected value (initially: `x=0`), then moves back to the subplot origin along the x-axis. The path-information `data_fill` is created the same way as before, but now we add it as a `svg.append("path")` that has the `fill` attribute, which - as the name implies - fills the region inside it with a solid color.

```
// Evaluate the normal pdf
var x = linspace(-3, 3, 201);
var pdf = standard_Gaussian_pdf(x);

svg.append("circle")
    .attr("r", 10 * height / 600)
    .attr("cx", xScale_pdf(0))
    .attr("cy", yScale_pdf(standard_Gaussian_pdf([0])))
    .attr("fill", "#666666") // "#c3e7f9"
    .attr("stroke-width", 5 * height / 600)
    .attr("id", "marker_pdf");

// Get the data for the path
var data_pdf = [];
for (i = 0; i < x.length; i++) {
    data_pdf.push({
        x: x[i],
        y: pdf[i]
    })
}


var valueline_pdf = d3.svg.line()
    .x(function(d) {
        return xScale_pdf(d.x);
    })
    .y(function(d) {
        return yScale_pdf(d.y);
    });

svg.append("path")
    .attr("d", valueline_pdf(data_pdf))
    .attr("fill", "none")
    .attr("stroke-width", 3 * height / 600)
    .attr("stroke", "#666666")
    .attr("id", "thatline")

var fill_x, fill_pdf
var xpos = 0;
fill_x = linspace(x_limits[0], xpos, 201);
fill_pdf = standard_Gaussian_pdf(fill_x);
fill_x.push(xpos)
fill_x.push(x_limits[0])
fill_pdf.push(0)
fill_pdf.push(0)
var data_fill = [];
for (i = 0; i < fill_x.length; i++) {
    data_fill.push({
        x: fill_x[i],
        y: fill_pdf[i]
    })
}
svg.append("path")
    .attr("d", valueline_pdf(data_fill))
    .attr("fill", "#999999")
    .attr("stroke", "None")
    .attr("id", "fill")
    .lower()
```

The next block essentially repeats the same procedure for the right subplot, but replaces the pdf evaluation with a cdf evaluation. We also add a dashed horizontal line that marks the

integral value on the y-axis. Go through the code and see if you recognize parallels to the code above.

```
// Define a subplot for the standard normal
const y_limits_cdf = [0, 1];
const window_x_cdf = [width * 0.585, width * 0.985];
const window_y_cdf = [height * 0.1, height * 0.89];

// Get scaling functions for the x scale and the y_scale
const xScale_cdf = d3.scaleLinear()
    .domain([x_limits[0], x_limits[1]])
    .range(window_x_cdf)
const xScale_cdf_inverse = d3.scaleLinear()
    .domain(window_x_cdf)
    .range([x_limits[0], x_limits[1]])
const yScale_cdf = d3.scaleLinear()
    .domain([y_limits_cdf[0], y_limits_cdf[1]])
    .range([window_y_cdf[1], window_y_cdf[0]])

// Draw the x axis
svg
    .append("g")
    .attr("transform", "translate(0," + window_y_cdf[1].toString() + ")")
    .call(d3.axisBottom(xScale_cdf).ticks(5))
    .style("font-size", (12 * width / 600).toString() + "px")
svg.append("text")
    .attr("transform",
        "translate(" + (math.mean(window_x_cdf)).toString() + "," + (height * 0.
    .style("text-anchor", "middle")
    .text("x")
    .style("font-family", "arial")
    .style("font-size", (12 * width / 600).toString() + "px")

// Draw the y axis
svg
    .append("g")
    .attr("transform", "translate(" + (window_x_cdf[0]).toString() + ",0)")
    .attr("id", "mainxaxis")
    .call(d3.axisLeft(yScale_cdf))
    .style("font-size", (12 * width / 600).toString() + "px");
//.call(d3.axisLeft(yScale).tickFormat(""));
svg.append("text")
    .attr("transform",
        "translate(" + (width * 0.525).toString() + "," + (math.mean(window_y_cd
    .style("text-anchor", "middle")
    .text("cumulative probability")
    .style("font-size", (12 * width / 600).toString() + "px")

// Draw the subplot label
svg.append("text")
    .attr("transform",
        "translate(" + ((window_x_cdf[1] - window_x_cdf[0])/2 + window_x_cdf[0])
    .style("text-anchor", "middle")
    .text("cdf")
    .style("font-size", (16 * width / 600).toString() + "px")

// Evaluate the normal pdf
var cdf = standard_Gaussian_cdf(x);

svg.append("circle")
    .attr("r", 10 * height / 600)
    .attr("cx", xScale_cdf(0))
    .attr("cy", yScale_cdf(standard_Gaussian_cdf([0])))
    .attr("fill", "#666666") // "#c3e7f9"
    .attr("stroke-width", 5 * height / 600)
    .attr("id", "marker_cdf");
```

```
        // Get the data for the path
        var data_cdf = [];
        for (i = 0; i < x.length; i++) {
            data_cdf.push({
                x: x[i],
                y: cdf[i]
            })
        }

        var valueline_cdf = d3.svg.line()
            .x(function(d) {
                return xScale_cdf(d.x);
            })
            .y(function(d) {
                return yScale_cdf(d.y);
            });

        svg.append("path")
            .attr("d", valueline_cdf(data_cdf))
            .attr("fill", "none")
            .attr("stroke-width", 3 * height / 600)
            .attr("stroke", "#666666") //"#4794c1")
            .attr("id", "thatline")

        svg.append('line')
          .attr('x1', window_x_cdf[0])
          .attr('x2', window_x_cdf[1])
          .attr('y1', yScale_cdf(standard_Gaussian_cdf([0])))
          .attr('y2', yScale_cdf(standard_Gaussian_cdf([0])))
          .attr('stroke', '#000000')
          .attr("stroke-width", 3 * height / 600)
          .style("stroke-dasharray", ((10 * height / 600).toString()+","+(5 * height / 6
          .attr("id","line_cdf")
          .lower();
```

So far, this element is purely static. It's time to introduce some interactivity! We start by defining three new variables: `movex` (x-position of the mouse), `movey` (y-position of the mouse), and `xpos` (what x-value we currently are looking at).

Make interactive elements with d3 is extremely easy. Here, for instance, we simply use the syntax `svg.on("mousemove")`, which calls a function whenever the mouse moves over the canvas. We then define an unnamed function that reads out the current coordinates of the cursor (`d3.event.x` and `d3.event.y`), then does whatever we want with it. For this element, we choose to only update the figure if the cursor's x-position is within one of the subplots. We can test this with a simple if-clause:

If the cursor's x-position is within the first subplot (i.e., `(movex >= window_x_pdf[0] && movex <= window_x_pdf[1]) == True`), then we convert the x-position to an x-value using the `xScale_pdf_inverse` function we created before. We then

- select the circular position markers for the pdf (`d3.select("#marker_pdf")`) and cdf (`d3.select("#marker_cdf")`) we created before via their IDs, and update their positions in

both subplots.

- Likewise, we update the fill path in the pdf plot (`d3.select("#fill")`) with the new `xpos`, and
- update the horizontal bar in the cdf subplot (`d3.select("#line_cdf")`). If our cursor is within the second subplot, the second else-if clause gets activated. This clause basically does the same thing as the one for the first subplot, but uses the helper function `xScale_cdf_inverse` instead, thereby ensuring that we update both subplots by hovering over either one.

One of the nice things about these interactive JavaScript elements is that you can make the updates to your elements as complex as you want, as long as you find a way to code it. The sky is the limit! In this simple example, we really just update the properties of pre-existing elements, but in principle you can also dynamically create new elements, destroy old elements, and so on.

Finally, we close all open environments, and are done with this simple interactive element!

```
// Shift the marker around on mouseover; restrict it to the contour
var movex, movey, xpos

svg
    .on("mousemove", function() {
        // Get the current mouseover coordinates
        movex = d3.event.x;
        movey = d3.event.y;


        if (movex >= window_x_pdf[0] && movex <= window_x_pdf[1]) {
               xpos = xScale_pdf_inverse(movex);

          d3.select("#marker_pdf")
              .attr("cx", xScale_pdf(xpos))
              .attr("cy", yScale_pdf(standard_Gaussian_pdf([xpos])));
          d3.select("#marker_cdf")
              .attr("cx", xScale_cdf(xpos))
              .attr("cy", yScale_cdf(standard_Gaussian_cdf([xpos])));

          fill_x = linspace(x_limits[0], xpos, 201);
          fill_pdf = standard_Gaussian_pdf(fill_x);
          fill_x.push(xpos)
          fill_x.push(x_limits[0])
          fill_pdf.push(0)
          fill_pdf.push(0)
          var data_fill = [];
          for (i = 0; i < fill_x.length; i++) {
              data_fill.push({
                  x: fill_x[i],
                  y: fill_pdf[i]
              })
          }
          d3.select("#fill")
              .attr("d", valueline_pdf(data_fill))
              .lower();


          d3.select("#line_cdf")
            .attr('y1', yScale_cdf(standard_Gaussian_cdf([xpos])))
            .attr('y2', yScale_cdf(standard_Gaussian_cdf([xpos])))
            .lower();

        } else if (movex >= window_x_cdf[0] && movex <= window_x_cdf[1]) {
               xpos = xScale_cdf_inverse(movex);
          d3.select("#marker_pdf")
              .attr("cx", xScale_pdf(xpos))
              .attr("cy", yScale_pdf(standard_Gaussian_pdf([xpos])));
          d3.select("#marker_cdf")
              .attr("cx", xScale_cdf(xpos))
              .attr("cy", yScale_cdf(standard_Gaussian_cdf([xpos])));

          fill_x = linspace(x_limits[0], xpos, 201);
          fill_pdf = standard_Gaussian_pdf(fill_x);
          fill_x.push(xpos)
          fill_x.push(x_limits[0])
          fill_pdf.push(0)
          fill_pdf.push(0)
          var data_fill = [];
          for (i = 0; i < fill_x.length; i++) {
              data_fill.push({
                  x: fill_x[i],
                  y: fill_pdf[i]
```

```
                    })
                }
                d3.select("#fill")
                    .attr("d", valueline_pdf(data_fill))
                d3.select("#fill_cdf")
                  .attr('y', yScale_cdf(standard_Gaussian_cdf([xpos])))
                  .attr('height', window_y_cdf[1] - yScale_cdf(standard_Gaussian_cdf([
                  .lower();

                d3.select("#line_cdf")
                  .attr('y1', yScale_cdf(standard_Gaussian_cdf([xpos])))
                  .attr('y2', yScale_cdf(standard_Gaussian_cdf([xpos])))
                  .lower();

            }

        });

    </script>
  </body>
</html>
```

HTML code for the pdf/cdf element

Here is the code for the element above in full:

```
<!DOCTYPE html>
<html>
   <head>
      <script src="https://unpkg.com/mathjs/lib/browser/math.js"></script>
      <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.17/d3.min.js"></script>
      <script src="https://d3js.org/d3.v4.min.js"></script>
      <style>
         body{
         margin: 0; overflow: hidden;
         font-family: Helvetica, sans-serif;
         }
      </style>
   </head>
   <!-- Create a div where the graph will take place -->
   <div id="my_datavisualization">
      <svg id="click" xmlns="http://www.w3.org/2000/svg">
      </svg>
   </div>
   <body>
      <script>
         // Get the viewport height and width
         const vw = Math.max(document.documentElement.clientWidth || 0, window.innerWidth
         const vh = Math.max(document.documentElement.clientHeight || 0, window.innerHeig

         // Fit to viewport
         var height = vw * 0.5;
         var width = vw;

         // Create the canvas. We will use only part of it for the main plot
         var svg = d3.select("#click") // This selects the div
             .attr("width", width) // This defines the canvas' width
             .attr("height", height) // This defines the canvas' height

         function linspace(start, end, resolution) {
             var spacing = [];
             // Go through a for-loop
             var i;
             for (i = 0; i < resolution; i++) {
                 spacing.push(start + (end - start) * i / (resolution - 1))
             }
             return spacing; // The function returns the linspace between p1 and p2
         }

         function standard_Gaussian_pdf(x) {
             mean = 0;
             std = 1;
             // Create a dummy variable
             var result = [];
             normalize = math.dotDivide(
                 1,
                 math.sqrt(
                     math.dotMultiply(
                         math.dotMultiply(
                             2,
                             math.PI),
                         math.pow(
                             std,
                             2))));

             var i;
             for (i = 0; i < x.length; i++) {
                 // Evaluate the first element of the Gaussian mixture
                 expon = math.exp(
```

```
                -math.dotDivide(
                    math.pow(
                        x[i] - mean,
                        2),
                    math.dotMultiply(
                        2,
                        math.pow(
                            std,
                            2))))
            temp = math.dotMultiply(
                normalize,
                expon)
            result.push(temp)
        }
        return result
    }

    function standard_Gaussian_cdf(x) {
        mean = 0;
        std = 1;
        // Create a dummy variable
        var result = [];
        for (i = 0; i < x.length; i++) {
            // Evaluate the first element of the Gaussian mixture
            temp = 0.5 * (1 + math.erf(x[i] /math.sqrt(2)))
            result.push(temp)
        }
        return result
    }

    // Define a subplot for the standard normal
    const x_limits = [-3, 3];
    const y_limits_pdf = [0, 1 / math.sqrt(2 * math.pi) * 1.2];

    const window_x_pdf = [width * 0.1, width * 0.5];
    const window_y_pdf = [height * 0.1, height * 0.89];

    // Get scaling functions for the x scale and the y_scale
    const xScale_pdf = d3.scaleLinear()
        .domain([x_limits[0], x_limits[1]])
        .range(window_x_pdf)
    const xScale_pdf_inverse = d3.scaleLinear()
        .domain(window_x_pdf)
        .range([x_limits[0], x_limits[1]])
    const yScale_pdf = d3.scaleLinear()
        .domain([y_limits_pdf[0], y_limits_pdf[1]])
        .range([window_y_pdf[1], window_y_pdf[0]])

    // Draw the x axis
    svg
        .append("g")
        .attr("transform", "translate(0," + window_y_pdf[1].toString() + ")")
        .call(d3.axisBottom(xScale_pdf).ticks(5))
        .style("font-size", (12 * width / 600).toString() + "px")
    svg.append("text")
        .attr("transform",
            "translate(" + (math.mean(window_x_pdf)).toString() + "," + (height * 0.
        .style("text-anchor", "middle")
        .text("x")
        .style("font-size", (12 * width / 600).toString() + "px")

    // Draw the y axis
    svg
        .append("g")
```

```
        .attr("transform", "translate(" + (window_x_pdf[0]).toString() + ",0)")
        .attr("id", "mainxaxis")
        .call(d3.axisLeft(yScale_pdf))
        .style("font-size", (12 * width / 600).toString() + "px");
    svg.append("text")
        .attr("transform",
            "translate(" + (width * 0.03).toString() + "," + (math.mean(window_y_pdf
        .style("text-anchor", "middle")
        .text("probability density")
        .style("font-size", (12 * width / 600).toString() + "px")

    // Draw the subplot label
    svg.append("text")
        .attr("transform",
            "translate(" + ((window_x_pdf[1] - window_x_pdf[0])/2 + window_x_pdf[0])
        .style("text-anchor", "middle")
        .text("pdf")
        .style("font-size", (16 * width / 600).toString() + "px")

    // Evaluate the normal pdf
    var x = linspace(-3, 3, 201);
    var pdf = standard_Gaussian_pdf(x);

    svg.append("circle")
        .attr("r", 10 * height / 600)
        .attr("cx", xScale_pdf(0))
        .attr("cy", yScale_pdf(standard_Gaussian_pdf([0])))
        .attr("fill", "#666666") // "#c3e7f9"
        .attr("stroke-width", 5 * height / 600)
        .attr("id", "marker_pdf");

    // Get the data for the path
    var data_pdf = [];
    for (i = 0; i < x.length; i++) {
        data_pdf.push({
            x: x[i],
            y: pdf[i]
        })
    }


    var valueline_pdf = d3.svg.line()
        .x(function(d) {
            return xScale_pdf(d.x);
        })
        .y(function(d) {
            return yScale_pdf(d.y);
        });

    svg.append("path")
        .attr("d", valueline_pdf(data_pdf))
        .attr("fill", "none")
        .attr("stroke-width", 3 * height / 600)
        .attr("stroke", "#666666") //"#4794c1")
        .attr("id", "thatline")

    var fill_x, fill_pdf
    var xpos = 0;
    fill_x = linspace(x_limits[0], xpos, 201);
    fill_pdf = standard_Gaussian_pdf(fill_x);
    fill_x.push(xpos)
    fill_x.push(x_limits[0])
    fill_pdf.push(0)
    fill_pdf.push(0)
```

```
var data_fill = [];
for (i = 0; i < fill_x.length; i++) {
    data_fill.push({
        x: fill_x[i],
        y: fill_pdf[i]
    })
}
svg.append("path")
    .attr("d", valueline_pdf(data_fill))
    .attr("fill", "#999999")
    .attr("stroke", "None") //"#4794c1")
    .attr("id", "fill")
    .lower()

// Define a subplot for the standard normal
const y_limits_cdf = [0, 1];
const window_x_cdf = [width * 0.585, width * 0.985];
const window_y_cdf = [height * 0.1, height * 0.89];

// Get scaling functions for the x scale and the y_scale
const xScale_cdf = d3.scaleLinear()
    .domain([x_limits[0], x_limits[1]])
    .range(window_x_cdf)
const xScale_cdf_inverse = d3.scaleLinear()
    .domain(window_x_cdf)
    .range([x_limits[0], x_limits[1]])
const yScale_cdf = d3.scaleLinear()
    .domain([y_limits_cdf[0], y_limits_cdf[1]])
    .range([window_y_cdf[1], window_y_cdf[0]])

// Draw the x axis
svg
    .append("g")
    .attr("transform", "translate(0," + window_y_cdf[1].toString() + ")")
    .call(d3.axisBottom(xScale_cdf).ticks(5))
    .style("font-size", (12 * width / 600).toString() + "px")
svg.append("text")
    .attr("transform",
        "translate(" + (math.mean(window_x_cdf)).toString() + "," + (height * 0.
    .style("text-anchor", "middle")
    .text("x")
    .style("font-family", "arial")
    .style("font-size", (12 * width / 600).toString() + "px")

// Draw the y axis
svg
    .append("g")
    .attr("transform", "translate(" + (window_x_cdf[0]).toString() + ",0)")
    .attr("id", "mainxaxis")
    .call(d3.axisLeft(yScale_cdf))
    .style("font-size", (12 * width / 600).toString() + "px");
//.call(d3.axisLeft(yScale).tickFormat(""));
svg.append("text")
    .attr("transform",
        "translate(" + (width * 0.525).toString() + "," + (math.mean(window_y_cd
    .style("text-anchor", "middle")
    .text("cumulative probability")
    .style("font-size", (12 * width / 600).toString() + "px")

// Draw the subplot label
svg.append("text")
    .attr("transform",
        "translate(" + ((window_x_cdf[1] - window_x_cdf[0])/2 + window_x_cdf[0])
    .style("text-anchor", "middle")
```

```
    .text("cdf")
    .style("font-size", (16 * width / 600).toString() + "px")

// Evaluate the normal pdf
var cdf = standard_Gaussian_cdf(x);

svg.append("circle")
    .attr("r", 10 * height / 600)
    .attr("cx", xScale_cdf(0))
    .attr("cy", yScale_cdf(standard_Gaussian_cdf([0])))
    .attr("fill", "#666666") // "#c3e7f9"
    .attr("stroke-width", 5 * height / 600)
    .attr("id", "marker_cdf");

// Get the data for the path
var data_cdf = [];
for (i = 0; i < x.length; i++) {
    data_cdf.push({
        x: x[i],
        y: cdf[i]
    })
}

var valueline_cdf = d3.svg.line()
    .x(function(d) {
        return xScale_cdf(d.x);
    })
    .y(function(d) {
        return yScale_cdf(d.y);
    });

svg.append("path")
    .attr("d", valueline_cdf(data_cdf))
    .attr("fill", "none")
    .attr("stroke-width", 3 * height / 600)
    .attr("stroke", "#666666") //"#4794c1")
    .attr("id", "thatline")

svg.append('line')
  .attr('x1', window_x_cdf[0])
  .attr('x2', window_x_cdf[1])
  .attr('y1', yScale_cdf(standard_Gaussian_cdf([0])))
  .attr('y2', yScale_cdf(standard_Gaussian_cdf([0])))
  .attr('stroke', '#000000')
  .attr("stroke-width", 3 * height / 600)
  .style("stroke-dasharray", ((10 * height / 600).toString()+","+(5 * height / 6
  .attr("id","line_cdf")
  .lower();


// Shift the marker around on mouseover; restrict it to the contour
var movex, movey, xpos

svg
    .on("mousemove", function() {
        // Get the current mouseover coordinates
        movex = d3.event.x;
        movey = d3.event.y;


        if (movex >= window_x_pdf[0] && movex <= window_x_pdf[1]) {
               xpos = xScale_pdf_inverse(movex);

          d3.select("#marker_pdf")
```

```
            .attr("cx", xScale_pdf(xpos))
            .attr("cy", yScale_pdf(standard_Gaussian_pdf([xpos])));
        d3.select("#marker_cdf")
            .attr("cx", xScale_cdf(xpos))
            .attr("cy", yScale_cdf(standard_Gaussian_cdf([xpos])));

        fill_x = linspace(x_limits[0], xpos, 201);
        fill_pdf = standard_Gaussian_pdf(fill_x);
        fill_x.push(xpos)
        fill_x.push(x_limits[0])
        fill_pdf.push(0)
        fill_pdf.push(0)
        var data_fill = [];
        for (i = 0; i < fill_x.length; i++) {
            data_fill.push({
                x: fill_x[i],
                y: fill_pdf[i]
            })
        }
        d3.select("#fill")
            .attr("d", valueline_pdf(data_fill))
            .lower();


        d3.select("#line_cdf")
          .attr('y1', yScale_cdf(standard_Gaussian_cdf([xpos])))
          .attr('y2', yScale_cdf(standard_Gaussian_cdf([xpos])))
          .lower();

    } else if (movex >= window_x_cdf[0] && movex <= window_x_cdf[1]) {
         xpos = xScale_cdf_inverse(movex);
        d3.select("#marker_pdf")
            .attr("cx", xScale_pdf(xpos))
            .attr("cy", yScale_pdf(standard_Gaussian_pdf([xpos])));
        d3.select("#marker_cdf")
            .attr("cx", xScale_cdf(xpos))
            .attr("cy", yScale_cdf(standard_Gaussian_cdf([xpos])));

        fill_x = linspace(x_limits[0], xpos, 201);
        fill_pdf = standard_Gaussian_pdf(fill_x);
        fill_x.push(xpos)
        fill_x.push(x_limits[0])
        fill_pdf.push(0)
        fill_pdf.push(0)
        var data_fill = [];
        for (i = 0; i < fill_x.length; i++) {
            data_fill.push({
                x: fill_x[i],
                y: fill_pdf[i]
            })
        }
        d3.select("#fill")
            .attr("d", valueline_pdf(data_fill))
        d3.select("#fill_cdf")
          .attr('y', yScale_cdf(standard_Gaussian_cdf([xpos])))
          .attr('height', window_y_cdf[1] - yScale_cdf(standard_Gaussian_cdf([
          .lower();

        d3.select("#line_cdf")
          .attr('y1', yScale_cdf(standard_Gaussian_cdf([xpos])))
          .attr('y2', yScale_cdf(standard_Gaussian_cdf([xpos])))
          .lower();

    }
```

```
            });

        </script>
    </body>
</html>
```

Interactive content: Run Python inside your book

Contents

Sphinx Extension

Our book has been enable to run Python code live in the browser (thanks Max!). This extension makes code cells in your `.ipynb` book pages editable and executable by the reader! Opposed to the original sphinx-thebe extensions, our extension runs python in the reader's browser and doesn't rely on an external webserver for the python kernel. We highly recommend this extension as it allows you to use the strength of combining text/figures and code of a jupyter book with interactivity!

This page contains some installation instructions and the other sections show how to use this functionality to create interactive figures and feedback on code

To see this happening, click 🚀 –> **Live Code** on the top right corner of this page

```
print('hello world')
```

Note

There are two tools presented on this page: Sphinx-Thebe and JupyterLite (at the bottom). Both tools rely on Pyodide, a software tool that is the backbone allowing for a Jupyter Notebook-like experience in a web browser without requiring the user to install any software.

Most of the TeachBooks Team uses a custom version of Sphinx-Thebe, a Sphinx Extension, because it can be integrated directly in a TeachBooks and provides a notebook-style experience (a "your page is a notebook" experience), whereas JupyterLite is more of a "notebook-in-a-window-in-your-page" experience.

This use of Sphinx-Thebe with Pyodide was made possible in Summer 2023 by Max Guichard, originally as a set of static files loaded in the MUDE Book. In January of 2024 this was converted to a custom Sphinx extension hosted on MUDE's TU Delft GitLab. As of February, 2025, we are currently working on improving this extension and providing it via PyPI, similar to our other tools. Stay tuned!

Sphinx-Thebe can be configured to use two types of "computers" in the background: Pyodide (what we use, running *100% in the browser*) and Binder, which is a third-party service that provides cloud-based computers for free (i.e., code runs on a separate server). This is a great tool, but often the time required to wait for the free cloud computer to be ready is too long (or simply does not work). As most of our teaching applications only require simple pieces of code, we prefer the Pyodide-based options because it is more reliable and loads faster.

Setting up Python live coding

To set up the Python live coding you need to add our own sphinx-thebe extension to your book. This extensions doesn't rely on a 3rd party like Binder and it supports local python execution and custom features. Therefore, you need to add some lines to `requirements.txt` and `_config.yml`

For `requirements.txt` add the following line:

```
git+https://github.com/TeachBooks/Sphinx-Thebe
```

This will download the correct version of the sphinx extension when the book is built (which loads the required packages from `requirements.txt`)

If you want to build the book locally, you need to install this version of sphinx-thebe in your environment as well with:

```
pip install -r requirements.txt
```

Afterwards, this sphinx extension needs to be enabled in your book. This can be done by adding the following lines to `_config.yml`:

```
launch_buttons:
  thebe: true

sphinx:
  config:
    html_js_files:
    - https://cdnjs.cloudflare.com/ajax/libs/require.js/2.3.4/require.min.js
    thebe_config:
      use_thebe_lite: true
      exclude_patterns: ["**/_*.yml", "**/*.md", "**/*.ipynb"]
```

The `html_js_files` link calls for some required javascript files. `use_thebe_lite` makes sure you initiate our adapted sphinx thebe extensions. The `exclude_patterns` makes sure you import all files so that they can be accessed from your code, except for the files matching the patterns shown.

> **Note**
>
> If you've a lot of big files which are not used by the code, the build book repository might grow a lot in size. Include unnecessary filetypes like images (i.e. `"**/*.png`) to `exclude_patterns`. Make sure you use proper markdown or MyST syntax for referencing to images. Using HTML code might lead to figures not being available in the final book. You can use this python script to list all filetypes in your book.

By `launch_buttons` you initiate the 🚀 –> **Live Code** on the top right corner of every page generated from a `.ipynb` file.

Note, if you're messing around with `html_theme_options` (for example when adding more buttons), all button behaviour is affected. In that case, as opposed to the jupyter book documentation, all buttons have to be specified from within `sphinx: html_theme_options` where you might also specify other buttons as well (ie. the code repository button).

```yaml
sphinx:
  config:
    html_js_files:
    - https://cdnjs.cloudflare.com/ajax/libs/require.js/2.3.4/require.min.js
    thebe_config:
      use_thebe_lite: true
      exclude_patterns: ["**/_*.yml", "**/*.md", "**/*.ipynb"]
    html_theme_options:
      repository_url: "hello!"
      use_repository_button: true
      launch_buttons:
        thebe: true
```

Instructions: local build

To test the fuctionality, you have to run a local server, otherwise the interactivity doens't work. So, you cannot just open the `index.html` file any more. This can be done with the steps shown in Local Server to view interactive elements locally.

If you push to main, you can test the interactivity on the book-draft website as well.

Custom cell tags

With the extended functionality of live code, additional cell tags have been developed to be added to python cells (not markdown cells) in any .ipynb file, next to the [existing tags] of the jupyter book (https://jupyterbook.org/en/stable/content/metadata.html):

- `disable-execution-cell` disables the ability to execute the cell when thebe is activated. This might be useful if you notebook file includes interactivity for only a part of the coding cells
- `disable-execution-page` disables the ability to execute the entire page. This might be useful if you use python code for creating a page, but the code is not part of the actual content of the book.
- `auto-execute-page` automatically starts the thebe live coding functionality whenever the page is opened. This is particularly useful if you're using widgets on a page.
- `thebe-init` directly starts running this cell as soon as the thebe live coding functionality starts.
- `thebe-remove-input-init`, as `thebe-init`, starts the cell directly as soon as the thebe live coding functionality starts. However, the input is not shown to the students (in static and interactive mode). The original `remove-input` tag will not work as it deletes the entire cell, so it can never be executed.

Additional packages

The python kernel doesn't have all packages standard included. Some of the most used packages which are included are:

- `python`
- `numpy`
- `scipy`
- `sympy`
- `matplotlib`
- `ipywidgets`

If you'd like to install more package, you can do so by added a codecell (preferably hidden using `thebe-remove-input-init`) with the following command:

```
import micropip
await micropip.install("package_name")
```

This will install the packages: Micropip will look at the Pyodide package index, but also at the general PyPi index. If a package is pure python (i.e. no C extensions), then it can also be used by Pyodide.

Custom packages

If you have some custom packages you can also create a wheel to import. Therefore, create a wheel from your package by running this in the root directoy of your package:

```
python setup.py bdist_wheel
```

Copy the wheel from `dist/<something>.whl` to your book and import the package from within a notebook using:

```
import micropip
await micropip.install('<directory>/<something>.whl')
```

Preferably with the tag `thebe-remove-input-init`.

Other packages not in Pyodide but on PyPI

If the packages are not included in Pyodide, you can use `pip:

```
%pip install package_name
```

Import module from subdirectory

In case you've a module in a subdirectory you cannot do directly `import <module_name> as <local_module_name>`. To solve this, you first have to add that subdirectory to the path with `sys.path.insert(1, '/<module_name/subdirectory>')` (preferably in a hidden cell with `theme-remove-input-init`)

Alternative with JupyterLite (less well integrated)

Another option to use python in your browser is to use JupyterLite. However, compared to Sphinx-Thebe, described above, this doesn't give a similar seamless interface within your page. Nevertheless, depending on your application, JupyterLite could be a useful tool, especially as it allows for three IDE types: Jupyter Notebook, Jupyter Lab and the iPython REPL.

One example use case is to create 'iframe-like' Ipython consoles / jupyterbook / jupyter lab environments in your book. As these 'iframes' are separate modules added to your page, the integration with the other content in your book is less flexible as with our sphinx-thebe extension. If you need to manage several different environments for JupyterLite within the book, you can use a JupyterLite session from a separate repo with the jupyter lab interface anyway, so you get the full browser experience. Alternatively, the ipython REPL better can be useful, as shown in our Python for Engineers book.

Tip

The TeachBooks Team has focused primarily on the use of Sphinx-Thebe and has not looked at JupyterLite over the last couple years. If you have an interesting application of this tool for teaching, please get in touch, we would love to see it! You can use GitHub Discussions or sent us an email at info@teachbooks.io.

Ipywidgets

Contents

If you publish widgets in a jupyter book they might (depending on your implementation) require an active python kernel for the output to be interactive. Using the thebe-integration, this is possible. Note: You can circumvent the use of a kernel by using packages which don't need such a kernel, for example the non-kernel-based widgets using Plotly.

`matplotlib` widgets

When you'd like to use `ipywidgets` in combinations with a `matplotlib` figure, import the following packages:

```
import ipywidgets as widgets
import micropip
await micropip.install("ipympl")
```

Activate the correct output format:

```
%matplotlib widget
```

Write the code which generates your plot in a function:

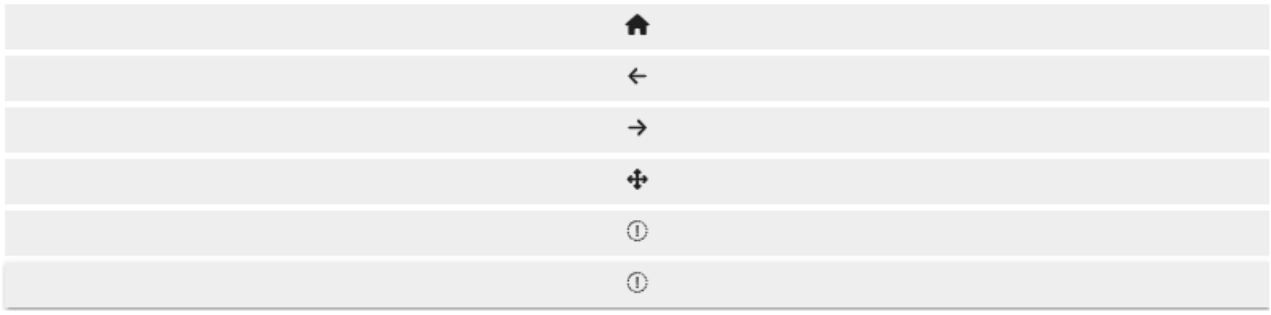
```
fig = plt.figure()  # Define the figure once
ax = fig.add_subplot(1,1,1)       # Define the axis once

def update_plot(input_variables_from_widget):
    ax.clear()   # Clear the existing plot
    ax.plot(x,y) # Plot your actual data
    plt.draw()   # Update the plot whenever this function is executed
```

Initiate the widget:

```
input_slider = widgets.FloatSlider(value=0, min=0, max=10, step=1, description='Input var
widgets.interact(update_plot, input_variable_from_widget = input_slider);
```

Finally, When using `%matplotlib widget`, there's the following toolbar:



Which only shows up when you hoover above. If you want to disable the toolbar you can use:

```
fig = plt.gcf()
fig.canvas.toolbar_visible = False
```

If you want it to be visible even if you don't hoover above it, use:

```
fig = plt.gcf()
fig.canvas.toolbar_visible = True
```

Please note:

- You might want to hide the output before the thebe has been activated.
- You can hide the code by adding a custom-made tag: `thebe-remove-input-init`.
- You can automatically start the thebe functionality when you add `auto-execute-page` to a cell.

Example

The code below shows an example of the strain energy in a structure and work done by a force for different trial functions. Because of the actual python code, the plot 'refreshes' upon updating a widgets. This can be avoided by updating the current figure instead of plotting a new figure. Remember, first press 🚀 –> **Live Code** and then wait until all cells are executed:

The graph will appear below in interactive mode:

Exercise checking using check-answer button

Contents

Using the python-enabled interactivity, exercise checking can be intuitively incorporated using a widgets which checks the value of certain variables with respect to the answer. These examples show different approaches to achieving this goal.

A function has been made to check the value of a variable. As input values it takes the variable name, the expected value (correct answer) and how these values should be compared. You need to add this function to each page, including the imports, and add the `thebe-remove-input-init` tag to the cell.

```
import ipywidgets as widgets
from IPython.display import display
import operator

def check_answer(variable_name, expected, comparison = operator.eq):
    output = widgets.Output()
    button = widgets.Button(description="Check answer")
    def _inner_check(button):
        with output:
            if comparison(globals()[variable_name], expected):
                output.outputs = [{'name': 'stdout', 'text': 'Correct!', 'output_type': '
            else:
                output.outputs = [{'name': 'stdout', 'text': 'Incorrect!', 'output_type':
    button.on_click(_inner_check)
    display(button, output)
```

To check a float value, you could use `math.isclose` as a function. For checking the equivalence of arrays, you might consider `numpy.array_equiv`

You can choose to include the correct answers in the same notebook. If students download the page and you didn't specify a custom url, they can find the correct answers. You can also place the correct answers in a seperate .py file which you import in the notebook.

Example

```
# This example has the user type in the answer as a Python variable, b
pi = 3.14
```

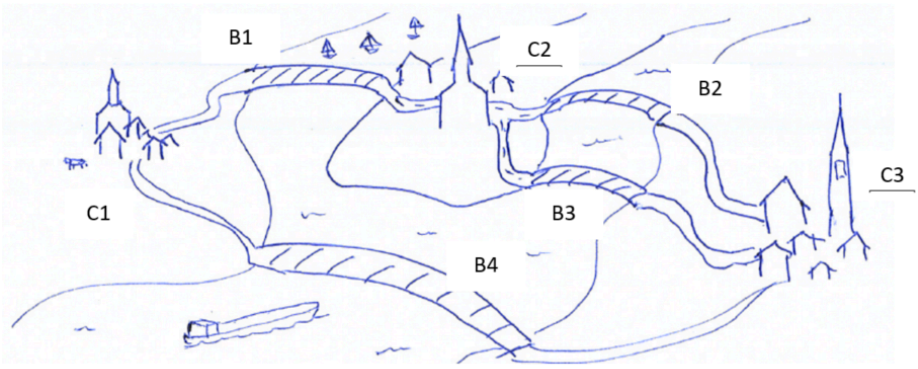
run run all add cell clear

JupyterQuiz

Contents

JupyterQuiz is a code-based implementation of multiple-choiche questions and numerical answer questions. This is a basic open course tool which requires you to write your quiz questions in Python language as a `dict` or to a `.json` file. An example is given below:

A pre-fabricated bridge design is being considered for river crossings in a remote region of the world, as shown in Figure 1. Cities 1, 2 and 3 are labelled C1, C2 and C3, with bridges labelled B1-B4.



JupyterQuiz

If the probability of failure for an individual bridge is 0.1 per year, compute the probability that you cannot drive from City 1 to City 2:

Type numeric answer here:

Iframes

Contents

This extension provides an interface to include iframes with relative ease, but does try to provide manners to interact with the various options. This rests purely by setting default CSS values, that the user can overwrite if preferred for individual iframes, but also globally. In general, each `iframe` is embedded within a `div` element, which eases sizing.

> **Note**
>
> Using CSS is complicated and error prone, so always check and never expect that you get what you want.

What does it do?

This extension provides several Sphinx directives:

- `iframe`
- `h5p`
- `video`
- `iframe-figure`

that can be used to quickly insert an iframe with standard sizing and styling.

Installation

To use this extenstion, follow these steps:

Step 1: Install the Package

Install the module `sphinx-iframes` package using `pip`:

```
pip install sphinx-iframes
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
sphinx-iframes
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions (**important**: underscore, not dash this time):

```
sphinx:
    extra_extensions:
        .
        .
        .
        - sphinx_iframes
        .
        .
        .
```

Configuration

The extension provides several configuration values, which can be added to `_config.yml`:

```
sphinx:
    config:
        -
        -
        -
        iframe_blend:          true # default value
        iframe_saturation:     1.5 # default value
        iframe_h5p_autoresize: true # default value
        iframe_background:     "#ffffff" # default value
        iframe_width:          calc(100% - 2.8rem) # default value
        iframe_aspectratio:    auto 2 / 1 # default value
        -
        -
        -
```

- `iframe_blend`: `true` (*default*) or `false`:
  - if `true` all iframes are standard blended with the background and in dark-mode also inverted.
  - if `false` all non-blended iframes will have background a colored background and no inversion for dark-mode is applied.
  - there's no need to set the blend or no-blend for individual iframes if it's set in the `_config.yml`, unless you want to deviate from the setting set there.
- `iframe_saturation`: `1.5` (*default*) or **float**:
  - Blended iframes are inverted in darkmode using the CSS filter `invert(1) hue-rotate(180deg) saturation(iframe_saturation)`.
- `iframe_h5p_autoresize`: `true` (*default*) or `false`:
  - if `true` all h5p iframes are automagically resized to fit the element in which the iframe is loaded.
  - if `false` no h5p iframes are automagically resized to fit the element in which the iframe is loaded.
- `iframe_background`: `"#ffffff"` (*default*) or **CSS string**:
  - sets the standard background color of non-blended iframes.
  - Any CSS string defining colors can be used, see CSS data type.
  - Surround with `" "` for hex strings.
  - Only visible if the content of the iframes has a transparant background.
- `iframe_width`: `calc(100% - 2.8rem)` (*default*) or **CSS string**:
  - sets the standard width of the iframe within the parent element;
  - Any CSS string defining a width can be used, see width CSS property.
- `iframe_aspectratio`: `auto 2 / 1` (*default*) or **CSS string**:

- sets the standard aspect ration of the iframe within the parent element;
- Any CSS string defining an aspect ratio can be used, see aspect-ratio CSS property.

Provided code

Directives

The following new directives are provided:

```
```{iframe} <link_to_webpage_to_embed>
```
```

```
```{h5p} <link_to_h5p_webpage_to_embed>
```
```

```
```{video} <link_to_video_to_embed>
```
```

In case of a YouTube-link, it is inverted to an embed link if the normal web URL is provided.

```
```{iframe-figure} <link_to_webpage_to_embed>
:name: some:label

The caption for the iframe.
```
```

Note that you don't need the full embed code of an iframe. Only the source url should be used.

All of these have the following options:

- `:class:`
  - If further CSS styling is needed, then use this option to append a CSS class name to the rendered iframe.
  - We recommend to only use the classes `blend` and `no-blend`, see Examples and details.
- `:width:`
  - Sets the width of the iframe. Use CSS compatible strings.
- `:height:`
  - Sets the width of the iframe. Use CSS compatible strings.
- `:aspectratio:`
  - Sets the width of the iframe. Use CSS compatible strings.
- `:styleframe:`
  - Sets the style of the iframe. Use CSS compatible strings. Surround with " ".
- `:stylediv:`
  - Sets the style of the surrounding div. Use CSS compatible strings. Surround with " ".

The directive `iframe-figure` also inherits all options from the `figure` directive from Sphinx.

Examples and details

`iframe` directive

To clearly show the blending and sizing, we showcase everthing in a general titled admonition.

Default behavior

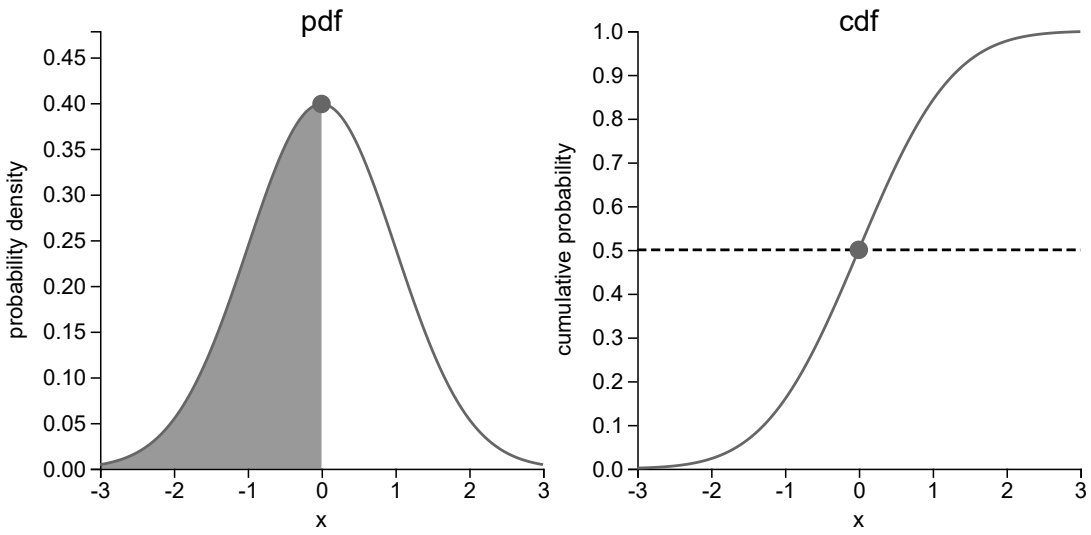
For use inline or in other directives and admonitions, iframes can be added using the following syntax:

````
```{iframe} <link_to_webpage_to_embed>
```
````

For example:

````
````{admonition} Default
```{iframe} ./some_content/element_pdf_and_cdf.html
```
````
````

Default

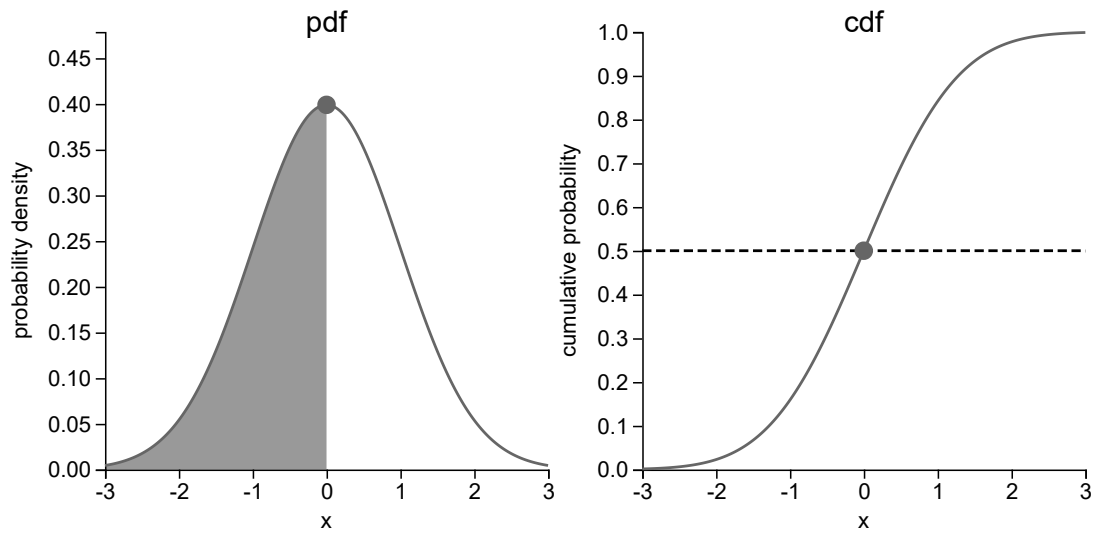


Blending

Blending can be enabled or disabled by using the classes `blend` and `no-blend`. Results may differ depending on other extensions and CSS code.

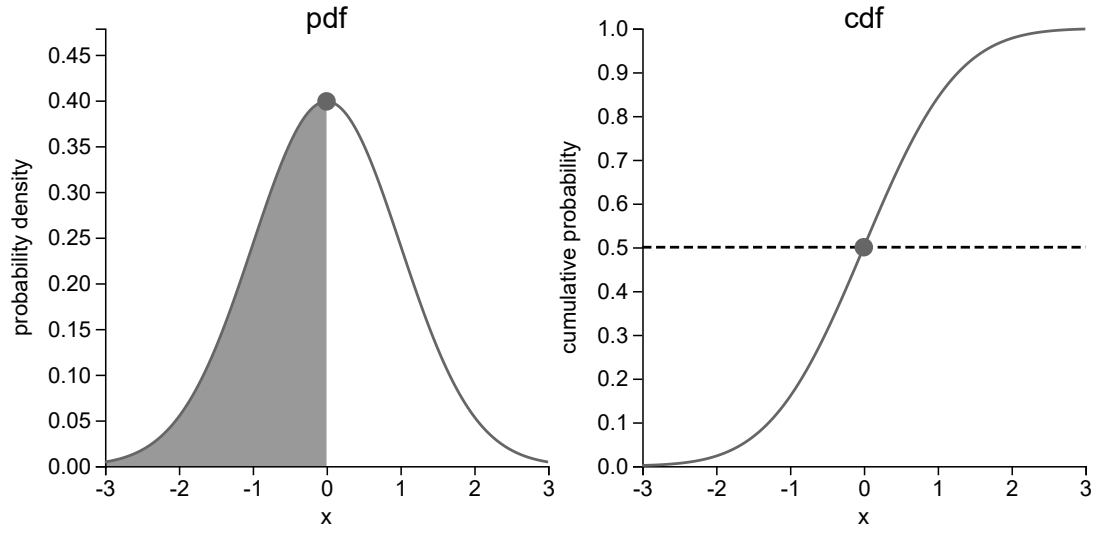
````
````{admonition} Enable blending
```{iframe} ./some_content/element_pdf_and_cdf.html
:class: blend
```
````
````

Enable blending




````
````{admonition} Disable blending
```{iframe} ./some_content/element_pdf_and_cdf.html
:class: no-blend
```
````
````

Disable blending



Sizing aspects

The size of the shown iframe can be controlled with atmost two out the following three options:

- `width`: Sets the width of the iframe. Use CSS compatible strings.
- `height`: Sets the height of the iframe. Use CSS compatible strings.
- `aspectratio`: Sets the aspect ratio of the iframe. Use CSS compatible strings.

These options will be applied to the encapsulating `div` element.

Note

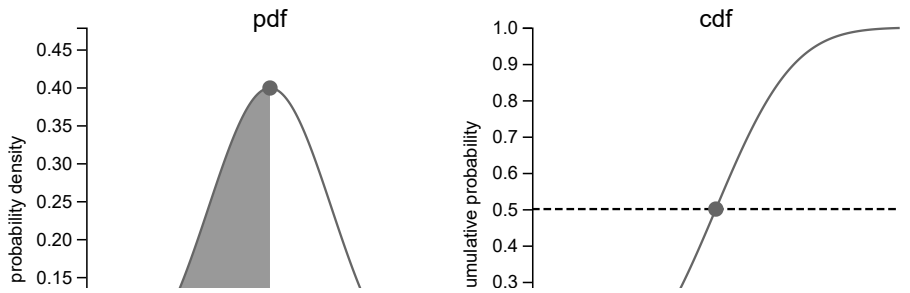
Using CSS is complicated and error prone, so always check and never expect that you get what you want.

Warning

This extension does not check the validity of the given options, nor checks whether at most two options are entered.

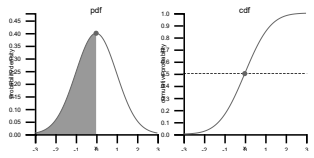
````
````{admonition} Width and height
```{iframe} ./some_content/element_pdf_and_cdf.html
:width: 600px
:height: 200px
```
````
````

Width and height



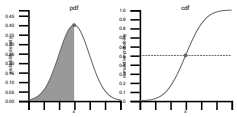

````
````{admonition} Width and aspect ratio
```{iframe} ./some_content/element_pdf_and_cdf.html
:width: 600px
:aspectratio: 2 / 3
```
````
````

Width and aspect ratio




````
````{admonition} Height and aspect ratio
```{iframe} ./some_content/element_pdf_and_cdf.html
:height: 150px
:aspectratio: 2 / 2
```
````
````

Height and aspect ratio



Styling aspects

The style of the shown iframe can be controlled with two options:

- `styleframe`: Sets the style of the iframe. Use CSS compatible strings. Include surround with `" "`.
- `styledivi`: Sets the style of surrounding div. Use CSS compatible strings. Include surround with `" "`.

Note

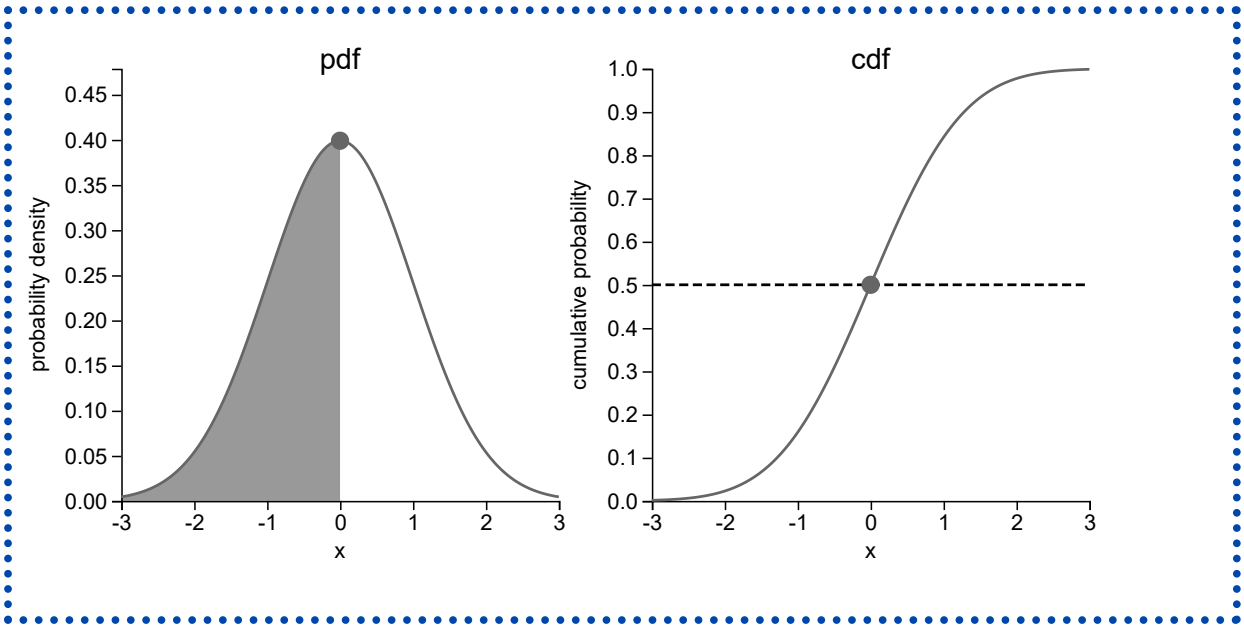
Using CSS is complicated and error prone, so always check and never expect that you get what you want.

Warning

This extension does not check the validity of the given option.

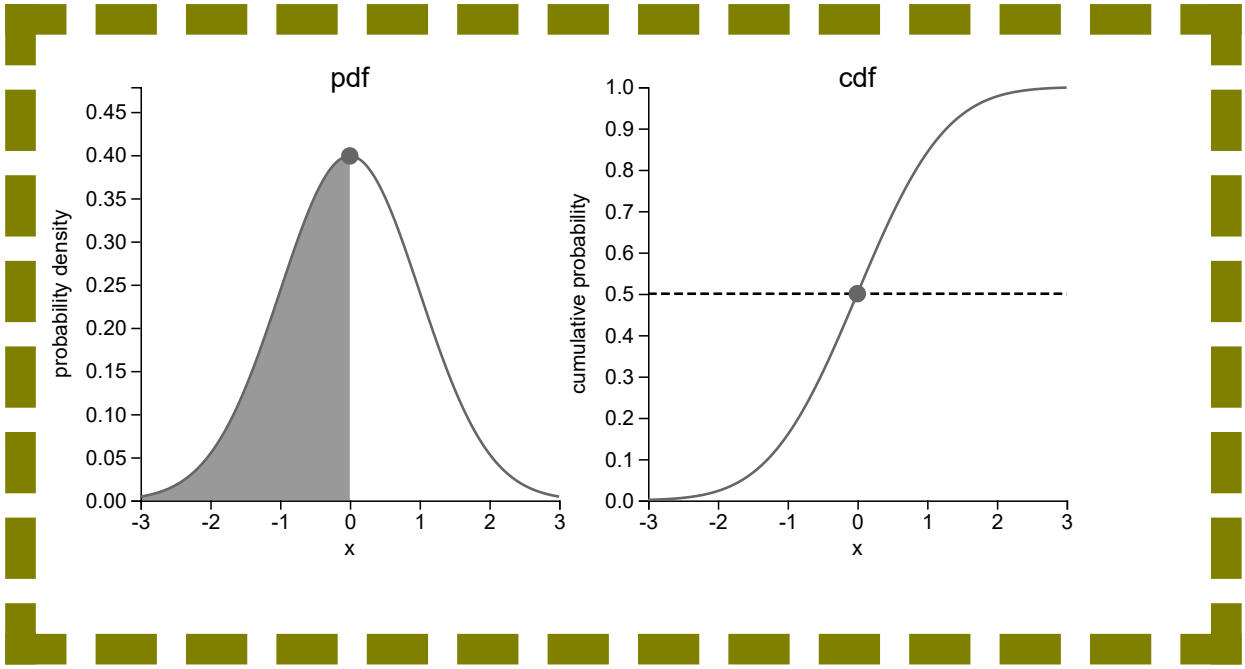
````
````{admonition} iframe styling
```{iframe} ./some_content/element_pdf_and_cdf.html
:styleframe: "border-style: dotted;border-color: #0047AB;border-width:5px;"
```
````
````

iframe styling



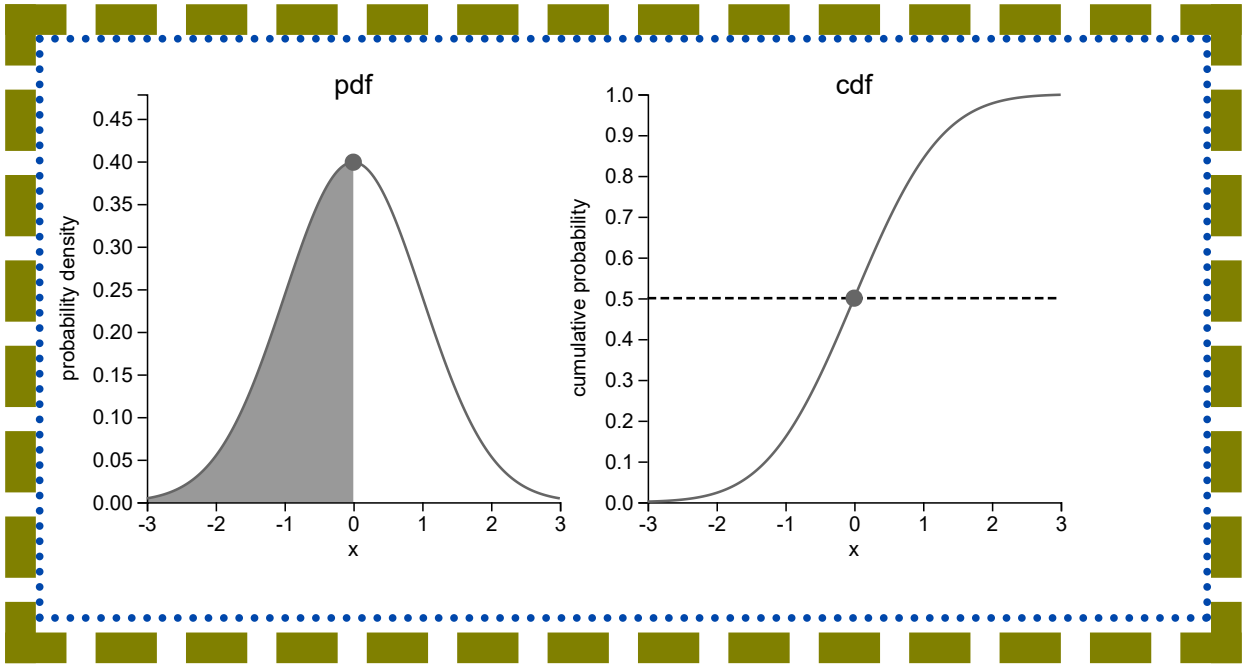
````
````{admonition} div styling
```{iframe} ./some_content/element_pdf_and_cdf.html
:stylediv: "border-style: dashed;border-color: olive;border-width:20px;"
```
````
````

div styling




````
````{admonition} iframe and div styling
```{iframe} ./some_content/element_pdf_and_cdf.html
:styleframe: "border-style: dotted;border-color: #0047AB;border-width:5px;"
:stylediv: "border-style: dashed;border-color: olive;border-width:20px;"
```
````
````

iframe and div styling



`h5p` directive

For iframes intended for H5P elements, the code

````
````{admonition} H5P example
```{iframe} https://tudelft.h5p.com/content/1292011179114024347/embed
:class: blend
:aspectratio: auto
```
````
````

can be reduced to

````
````{admonition} H5P example
```{h5p} https://tudelft.h5p.com/content/1292011179114024347/embed
```
````
````

resulting in

H5P example

If you want to run the command `numpy.linspace(0,100,100)` after importing as shown above, what is the code you need to enter?

Check

Note that you don't need the full embed code as provided by H5P. Only the source url (often with the following syntax `https://<h5p_host_server>/content/<h5p_element_id>/embed`) should be used. This url can be obtained from the url in your H5P application with an addtional `/embed`, or in the html-embed-code.

`video` directive

For iframes intended for videos, the code

````
````{admonition} video example
```{iframe} https://www.youtube.com/embed/B1J6Ou4q8vE?si=XZDT83fcR6W3Dxut
:class: no-blend
:styleframe: "background: transparent;"
:aspectratio: auto 16 / 9
```
````
````

can be reduced to

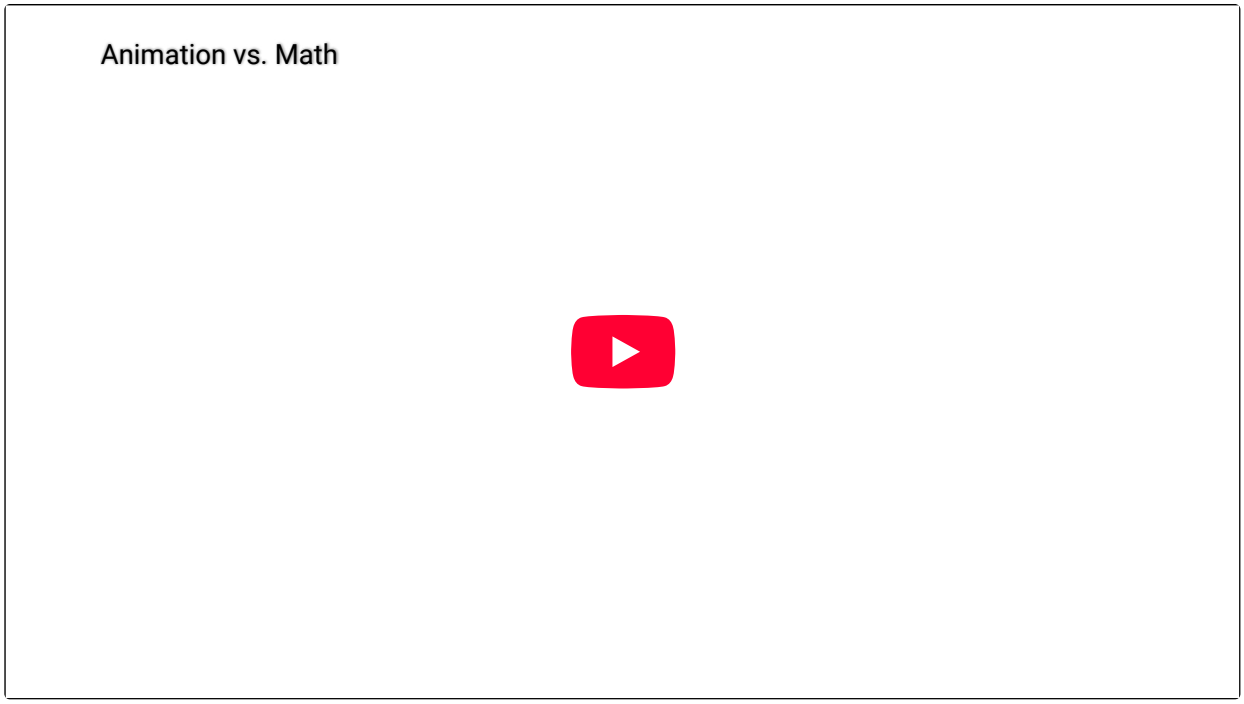
````
````{admonition} video example
```{video} https://www.youtube.com/embed/B1J6Ou4q8vE?si=XZDT83fcR6W3Dxut
```
````
````

or it can take the regular YouTube URL:

````
````{admonition} video example
```{video} https://www.youtube.com/watch/B1J6Ou4q8vE
```
````
````

resulting both in

video example



`iframe-figure` directive

In Fig. 115 you can find the result of the below code. The reference is made using `{numref}` and label behind `:name:`.

````
````{iframe-figure} ./some_content/element_pdf_and_cdf.html
:name: some:label

The caption for the iframe.
````
````

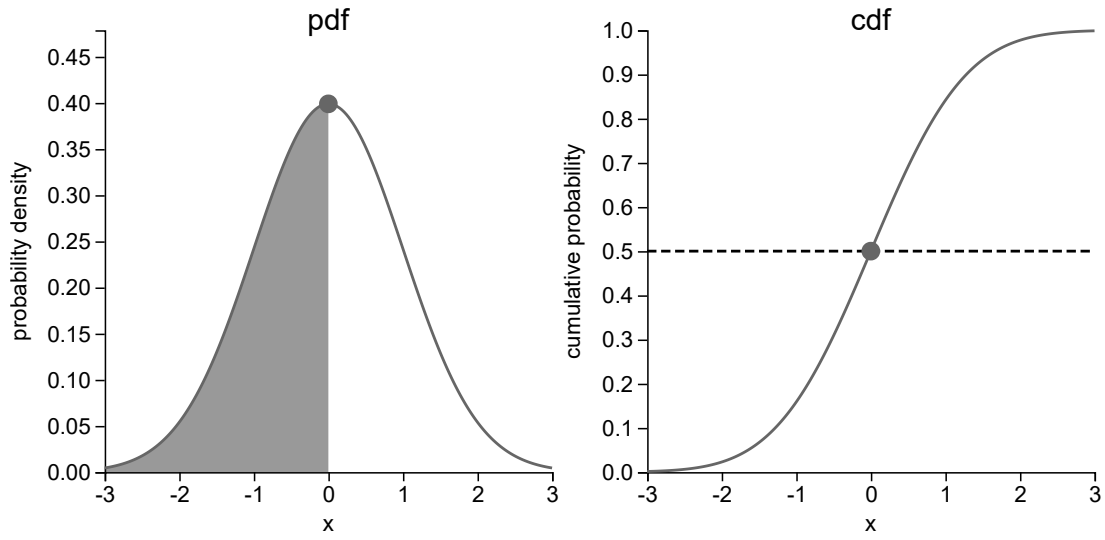


Fig. 115 The caption for the iframe.

Contribute

This tool's repository is stored on GitHub. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository.

The `README.md` of the branch `Manual` is also part of the TeachBooks manual as a submodule.

PRIME applets

Contents

This extension provides an interface to include PRIME applets with relative ease.

What does it do?

This extension provides one Sphinx directives (`applet`) that can be used to quickly insert a PRIME applet.

Installation

To use this extenstion, follow these steps:

Step 1: Install the Package

Install the module `sphinx-prime-applets` package using `pip`:

```
pip install sphinx-prime-applets
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
sphinx-prime-applets
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions (**important**: underscore, not dash this time):

```
sphinx:
    extra_extensions:
        .
        .
        .
        - sphinx_prime_applets
        .
        .
        .
```

Applet directive

````
```{applet}
:url: lines_and_planes/normal_equation_plane_origin
:fig: Images/image_shown_in_print_version.svg
:name: name_that_is_used_to_refer_to_this_figure
:class: dark-light
:title: This title is shown when you full-screen the applet

A plane through the point $P$.
```
````

> **Note**
>
> The `url` parameter should be the part of the URL after `/applet/`. So if the full URL is `https://openla.ewi.tudelft.nl/applet/lines_and_planes/normal_equation_plane_origin`, you should set the parameter to `lines_and_planes/normal_equation_plane_origin`.

Parameters for an applet

Some parameters can be set for an applet. Only the `url`, `fig` and `name` parameters are required; the rest is optional. It is recommended to add a `status` to the applet, which can be `unreviewed`, `in-review` or `reviewed`.

````
```{applet}
:url: lines_and_planes/normal_equation_plane_origin # Required url
:fig: Images/lines_and_planes/normal_equation_plane_origin.svg  # Image shown in print version
:status: reviewed # default is "unreviewed". Other options are "in-review" and "reviewed"
:name: Fig:InnerProduct:ProjectionVectorLine

A title that describes the applet
```
````

Optional parameters

Parameter	Description	Default
`title`	A string that will be shown as the title of the applet when the applet is in fullscreen mode	""
`status`	The status of the applet. Can be `unreviewed`, `in-review` or `reviewed`	`unreviewed`
`width`	The width of the applet in pixels	100%
`height`	The height of the applet in pixels	400px

Control parameters

> **Warning**
>
> Work in progress

2D Specific parameters

> **Tip**
>
> You should add split-* before the parameter to make it apply to the right scene

Parameter	Description	Default
position2D	The position of the applet in the 2D plane	0,0
zoom2D	The zoom level of the applet in the 2D plane	1

3D Specific parameters

> **Tip**
>
> You should add split-* before the parameter to make it apply to the right scene

Parameter	Description	Default
position3D	The position of the applet in the 3D plane	0,0,0
zoom3D	The zoom level of the applet in the 3D plane	1

Contribute

This tool's repository is stored on GitHub. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository.

The `README.md` of the branch `manual` is also part of the TeachBooks manual as a submodule.

Book styling

TeachBooks developed a few improvements of the book which improve the visual book styling.

Image Inverter

Contents

When toggling dark mode in JupyterBook, images and figures are not inverted by default, but a white background is inserted. However, this white background might not always be desired in dark mode.

The **Sphinx-Image-Inverter** extension provides a solution by applying an automatic filter to images and iframes. If this filter is not desired for certain items, the **Sphinx-Image-Inverter** extension provides a solution by allowing selective disabling using the `dark-light` class.

If the filter should *only* be applied to a small number of images, this can be done by applying the filter *only* to items with the `dark-light` class, in combination with setting `inverter_all` to `true` in `_config.yml`.

> **Note**
>
> The inversion does not apply to the logo. If a different logo is preferred in dark mode compared to light mode, please use Different logos for light and dark mode.

How does it work?

This Sphinx extension applies a filter such that dark and light colors are switched, however keeps the colours recognizable. This is particularly useful for graphs in which a certain colour is mentioned in accompanying text. Items are not converted if they are marked with the `dark-light` class (recommended for example for photos).

In more detail, first the colors of the element are inverted, then the hue of the colors is shifted by 180 degrees, so the inverted colors change to their complementary hues. This flips the brightness and contrast, while keeping the hue somewhat recognizable (so a blue line will be a blue line in both ligth and dark mode). Black and white stay inverted (so white becomes black, and black becomes white), because they don't have a hue. Next, the colors are (by default) saturated to enforce a better contrast. After this, the element blends with the background, making similar colors appear dark and very different colors appear bright. The overall effect creates high contrast between the element and the background, depending on their colors.

Installation

To install the Sphinx-Image-Inverter, follow these steps:

Step 1: Install the Package

Install the `sphinx-image-inverter` package using `pip`:

```
pip install sphinx-image-inverter
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
sphinx-image-inverter
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions:

```
sphinx:
    extra_extensions:
        - sphinx_image_inverter
```

Usage

Enable/Disable Inversion of all Images/Figures

By default all images and figures will be inverted. If this is wished for, use the following in your `_config.yml`:

```
sphinx:
    config:
        inverter_all: false
```

This stops automatic inversion of images and figures. Inversion of specific images and figures can be achieved by enabling this using the `dark-light` class, see below.

Disable/change saturation

The saturation level is preset to `1.5`. If no saturation or a different saturation is requested, use the following in your `_config.yml`:

```
sphinx:
    config:
        inverter_saturation: <saturation>
```

where `<saturation>` should be replace with a positive number. The value `1.0` represent no saturation and the value `1.5` is the default value.

Disable/Enable Image/Figure Inversion

By default, when dark-mode is toggled in JupyterBook, all images and figures are inverted. To prevent certain images from being inverted, apply the `dark-light` class. The steps for both Markdown and HTML formats are given below.

For Markdown Format

1. Locate the markdown file that contains the image or figure you want to exclude from inversion.
2. Add the `:class: dark-light` attribute to the figure directive.

   Example:

```
```{figure} example_folder/example_image.jpg
:class: dark-light
:width: 400```
```

For HTML Format

If your image or figure is defined using HTML, apply the `dark-light` class directly to the tag.

```
<iframe
    src="some_figure.html"
    width="600"
    height="300"
    class="dark-light">
</iframe>
```

Now your image will not be inverted when dark mode is toggled (in the default scenario).

If `inverter_all` has been set to `false`, only the image *with* the `dark-light` class will be inverted.

Display Text According to Theme

You may want to display different text depending on whether the dark mode or light mode is enabled. To do that, you can use the following classes:

- **Light Mode only:**

```
<span class="only-light">Text only visible in Light Mode.</span>
```

- **Dark Mode only:**

```
<span class="only-dark">Text only visible in Dark Mode.</span>
```

These classes make sure that your text is only visible in the specified modes.

Compatible LaTeX colours

If you'd like to use LaTeX colours which invert similarly, use the approach Sphinx extension Sphinx-Named-Colors.

Contribute

This tool's repository is stored on GitHub. The `README.md` of the branch `Manual` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.
```

JupyterBook-Patches

Contents

This Sphinx extension fixes:

- an issue where drop down menus would still take up space after being minimized, and the patch fixes it through some css.
- an issue where in drop down code cells the shown summary remained lightgray instead of turning darkgrey. Fix through css.
- an issue where the size of code in a header is not the correct font size. Fix through css.
- an issue where two buttons for interactive matplotlib widget do not appear.
- an issue where the sidebar shows a scrollbar even if that's not needed

Installation

To install the Sphinx-JupyterBook-Patches, follow these steps:

Step 1: Install the Package

Install the `jupyterbook_patches` package using `pip`:

```
pip install jupyterbook_patches
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
jupyterbook_patches
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions:

```
sphinx:
    extra_extensions:
        - jupyterbook_patches
```

Contribute

This tool's repository is stored on GitHub. The `README.md` of the branch `manual_docs` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository

of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

TU Delft theme

Contents

The default theme in JupyterBooks is usually not desired and need to be changed by adding custom stylesheets. The **Sphinx-TUDelft-theme** extension provides a simple solution to have a uniform theme across all the books created at Delft University of Technology that matches the TU Delft identity.

What does it do?

This extension applies styling changes, being

- particular colours (different colors for light and dark themes) for:
  - admonitions (e.g. hint, note, tip, error, etc.),
  - proofs (e.g. theorem, axiom, lemma, corollary, etc.),
  - exercises,
  - buttons,
  - badges,
  - custom components,
  - $\LaTeX$,
  - the primary and secondary color of the book (mainly used for typesetting links).
- particular icons for:
  - proofs (e.g. theorem, axiom, lemma, corollary, etc.),
  - exercises,
  - custom components.

Unless specified otherwise, see Usage, this extension also automatically sets:

- a Delft University of Technology logo;
- a Delft University of Technology favicon;
- the Delft University of Technology preferred fonts;
- rendering text inside MathJax as the surrounding text;
- an always visible logo (i.e. a sticky logo).

You can see how the TU Delft theme looks like applied in this example book.

Installation

To implement the TU Delft theme, follow these steps:

Step 1: Install the Package

Install the `sphinx-tudelft-theme` package using `pip`:

```
pip install sphinx-tudelft-theme
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
sphinx-tudelft-theme
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions (**important**: underscore, not dash this time):

```
sphinx:
    extra_extensions:
        .
        .
        .
        - sphinx_tudelft_theme
        .
        .
        .
```

The following Sphinx extra extensions (if used) must be added *before* this extension:

- `sphinx_proof`
- `sphinx_exercise`
- `teachbooks_sphinx_grasple`

If this is forgotten, the CSS of this extension cannot be applied correctly.

Usage

By following the steps above, the theme will be applied automatically.

To use the defined colors inside $\LaTeX$ rendered with MathJax, one should use the command

```
\class{<color>}{<math>}
```

where `<color>` is one of the following colors:

- tud-red
- tud-blue
- tud-green
- tud-raspberry
- tud-yellow
- tud-darkGreen
- tud-orange
- tud-cyan
- tud-gray

- tud-purple
- tud-pink
- tud-darkBlue

and `<math>` is the $LaTeX$ that should be rendered in the color `<color>`.

If a Delft University of Technology logo should not be set (i.e. use logos defined by the user), include the following in your `_config.yml` file:

```
sphinx:
  config:
    ...
    tud_change_logo: false
```

If a Delft University of Technology favicon should not be set (i.e. use a favicon defined by the user), include the following in your `_config.yml` file:

```
sphinx:
  config:
    ...
    tud_change_favicon: false
```

If the Delft University of Technology fonts should not be set (i.e. use fonts defined by the user), include the following in your `_config.yml` file:

```
sphinx:
  config:
    ...
    tud_change_fonts: false
```

If rendering text inside MathJax should not be the same as the surrounding html, include the following in your `_config.yml` file:

```
sphinx:
  config:
    ...
    tud_change_mtext: false
```

If a sticky logo is not preferred, include the following in your `_config.yml` file:

```
sphinx:
  config:
    ...
    tud_sticky_logo: false
```

Contribute

This tool's repository is stored on GitHub. The `README.md` of the branch `Manual` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

Rich hover over tips

Contents

Sphinx-tippy allows you to create rich hover over tips as demonstrated here: https://sphinx-tippy.readthedocs.io/en/latest/. This TeachBooks Tippy extension makes it plug-and-play within a TeachBook.

The differences with Sphinx-tippy are:

- Default inclusion of useful CSS-file.
- Default activation of sphinx.ext.mathjax.
- Default loading method of *defer* changed to *None* for supporting JavaScript.
- Default support for TeachBooks Sphinx Grasple such that images are shown (as iframes are not loaded).
- Limit the showing of tool tips to the `<main>` anchor of the Sphinx book.
- Limit the maximum height of a tool tip.
- Force tool tips to remain with visible document body.
- No arrow that points from the tool tip to the link.

What does it do?

You can see how the this works in the example book.

Installation

To install TeachBooks-Sphinx-Tippy, follow these steps:

Step 1: Install the Package

Install the `teachbooks-sphinx-tippy` package using `pip`:

```
pip install teachbooks-sphinx-tippy
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
teachbooks-sphinx-tippy
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions (**important**: underscore, not dash this time):

```
sphinx:
    extra_extensions:
        - teachbooks_sphinx_tippy
```

Usage

By following the steps above, the extension will be added automatically.

Contribute

This tool's repository is stored on GitHub. The `README.md` of the branch `Manual` is also part of the TeachBooks manual as a submodule. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository. To update the `README.md` shown in the TeachBooks manual, create a fork and open a merge request for the GitHub repository of the manual. If you intent to clone the manual including its submodules, clone using: `git clone --recurse-submodulesgit@github.com:TeachBooks/manual.git`.

APA References

Contents

> **Note**
>
> This feature is stable but because it relies on a local extension that is part of your book files and *not* managed as part of a computing environment (e.g., `pip` or `conda`), unknown issues may arise. We would like to convert this feature into an independent Sphinx extension and/or Pybtext plugin. Visit the project page on GitHub to learn more.
>
> The instructions provided here will work under conventional usage with Jupyter Book (e.g., `jupyter book build book`).
>
> A temporary fix has also been implemented in the TeachBooks package for use with the Deploy Book Workflow (release mode) (see release notes from v0.1.0).

Introduction

Do you include references in your book, but you're tired of the default options available in Jupyter Book? For example, the square-bracket style of citation and reference, where the first three letters of the name year are combined in a way that seems designed to minimize transparency?

There is a solution! This extension allows you to have APA formatting in your book.

Installation

To use APA-references, a set of Python files need to be manually loaded into your book

Step 1: files to root directory of your book

Download this zip-file `_ext.zip`, which contains the necessary files, unzip it and place the contents in the root of your book directory so that it looks like the schematic here:

```
book
├── _ext/
│   ├── pybtexapastyle/
│   ├── apastyle.py
│   ├── bracket_citation_style.py
├── _config.yml
├── _toc.yml
├── requirements.txt
├── <book files>
```

Step 2: Add to `requirements.txt`

Add a specific version of `docutils` to your requirements file:

```
docutils==0.17.1
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add a `bibtex_default_style`, `bybtex_reference_style` and the local extensions `apastyle` and `bracket_citation_style.py`

```
sphinx:
  config:
    ...
    bibtex_default_style: myapastyle
    bibtex_reference_style: author_year_round
  local_extensions:
    apastyle: _ext/
    bracket_citation_style: _ext/
  ...
```

Usage

All references are now made in APA-style. See for example this reference: Moore (2023) which shows up on the references page too. The form of the citation looks like this:

```
{cite:t}`jason_moore`
```

Here are three examples for making citations:

Syntax	Result
`{cite:t}`	Moore (2023)
`{cite:p}`	(Moore, 2023)
`{cite}`	(Moore, 2023)

For more options on the in-line citation style, see https://jupyterbook.org/en/stable/content/citations.html#change-the-in-line-citation-style.

Known Issues

Two issues are known:

- During the build, warning will be raised with `... WARNING: duplicate label for keys ...`. In most cases, these warnings can be ignored. As noted above, the book will not build references properly

- In case you have both an `.ipynb` and `.md` version of a file, the `.md` version will always be used whenever this extension is used. This removes the possibility to show code outputs.

Implementation

The extension is based on `pybtex`, which is a BibTeX-compatible bibliography processor in Python that is extendible with plugins.

Although some customization is possible with the standard Jupyter Book features, as described here, this extension implements the complete APA style, as well as enforcing round brackets (like this).

The need to enforce `docutils==0.17.1` version is the only known solution to the issue where empty brackets `[]` are left on the references page.

Contribute

This tool needs to be developed further to make it into a proper sphinx-extension (and/or an independent `pybtext` plugin). The process is described in this project page on GitHub. If you've ideas or questions, please reach out to us at info@teachbooks.io!

Custom named colors

Contents

This extensions provides a simple solution to use CSS named colors and ***custom named colors*** in:

- $\LaTeX$;
- MarkDown text;
- Admonitions.

What does it do?

This extension defines, based on the CSS named color and custom named colors (provided by the user), several new

- $\LaTeX$ commands;
- Sphinx roles;
- Sphinx admonitions;
- Sphinx admonition classes;

that are styled by a generated CSS file.

If specified, each color will have a different value in the light and dark data-theme.

Installation

To use this extenstion, follow these steps:

Step 1: Install the Package

Install the `sphinx-named-colors` package using `pip`:

```
pip install sphinx-named-colors
```

Step 2: Add to `requirements.txt`

Make sure that the package is included in your project's `requirements.txt` to track the dependency:

```
sphinx-named-colors
```

Step 3: Enable in `_config.yml`

In your `_config.yml` file, add the extension to the list of Sphinx extra extensions (**important**: underscore, not dash this time):

```
sphinx:
    extra_extensions:
        .
        .
        .
        - sphinx_named_colors
        .
        .
        .
```

Configuration

This extension provides some configuration values, which can be added to:

```
sphinx:
    config:
        .
        .
        .
        named_colors_include_CSS: true # default value
        named_colors_dark_and_light: true # default value
        named_colors_saturation: 1.5 # default value
        named_colors_custom_colors: None
        .
        .
        .
```

```
named_colors_include_CSS: true # default value
```

- If set to *true* all CSS named colors will be included in the extension.
- If set to *false* no CSS named colors will be included in the extension. If no custom named colors are defined, this extension will do nothing.

```
named_colors_dark_and_light: true # default value
```

- *true*: for all CSS named colors and all custom named colors a secondary value will be generated for use in the dark data-theme, unless otherwise specifed for custom colors. The generated colors emulate the same as the CSS filter `invert(1) hue_rotate(180) saturate(<val>);` where `<val>` is the value set by `named_colors_saturation`. This filter is also used in the Sphinx Image Inverter
- *false*: This disables the use of different colors in the dark data-theme, even if specified for custom colors.

```
named_colors_saturation: 1.5 # default value
```

- *number*: The saturation value used in the generation of the dark data-theme colors.

```
named_colors_custom_colors: None
```

- *None*: No custom named colors will be included.
- *dictionary*: A Python dictionary where each `key` defines a custom name and the `value` is a list of 3 or 6 integers, with each integer at minimum 0 and at maximum 255.
    - If 3 integers are provided, these are the RGB values of the custom named color and, if specified, the dark data-theme color will be generated.

- If 6 integers are provided, the first set of 3 integers form the RGB values of the custom named color and the second set of 3 integers form the RGB values of the dark data-theme color.
- Each key should contain only characters from the ranges `a-z`. Hyphens (`-`) are allowed, however this is not recommended.
- An example value:
  - `{'onlylight':[165,21,160],'lightanddark':[45,180,117,204,158,110]}`

Provided code

> **Note**
>
> In the next part, replace `namedcolor` by the name of the CSS/custom named color.

$\LaTeX$ elements

Named colors without hyphens

```
\namedcolor{...}
```

- Only use in $\LaTeX$ code.
- This will typeset `...` in the color *namedcolor*.

Named colors with hyphens

```
\class{namedcolor}{...}
```

- Only use in $\LaTeX$ code.
- This will typeset `...` in the color *namedcolor*.

MarkDown elements

```
{namedcolor}`...`
```

- Only use in *MarkDown* code.
- This will typeset `...` in the color *namedcolor*.

To provide the use of **strong** and/or *emphasis* colored text, we als provide the next three roles:

```
{namedcolor_strong}`...`
```

```
{namedcolor_emphasis}`...`
```

```
{namedcolor_strong_emphasis}`...`
```

These extra roles have been created using the extension sphinxnotes-comboroles.

Admonitions

Colored admonitions can be generated in two ways, explained below.

1. By adding a class to an existing admonition

```
::::{type} Title (optional or required, depending on type)
:class: namedcolor
Content
::::
```

2. By using a new admonition

```
::::{namedcolor} Title (optional)
Content
::::
```

If the title is omitted in the new admonition, the title bar will not be displayed.

In both cases extra classes can be added to the admonition to apply other styling.

A special new class for existing admonitions is also introduced: `no-title`. This suppresses printing of the title bar, even if the title is given. For the named color admonitions this happens automatically if no title is given.

For the named color admonitions the class `show-bar` is introduced for titleless admonitions. This forces printing of the title bar. If a title is given, the title will be printed too and adding the class `show-bar` is redundant.

> **Warning**
>
> Note that, because of the use of CSS, sometimes results may differ from the expected result.

Examples & details

Overview of chosen options for the examples

```
sphinx:
  config:
    named_colors_dark_and_light: true # default value
    named_colors_saturation: 1.5 # default value
    named_colors_include_CSS: true # default value
    named_colors_custom_colors: {'onlylight':[165,21,160],'lightanddark':[45,180,117,204,158,110],'hyphen-color':[45,180
```

LaTeX colors

Some examples of CSS named colors and the **custom named colors** used within LaTeX code. Do not forget to check out the colors in the dark data-theme!

Color	$\LaTeX$ Code	Result
olive	`\olive{\int_a^bf(x)dx}`	$\olive{\int_a^bf(x)dx}$
hotpink	`1.\hotpink{\mathbf{49}}`	$1.\hotpink{\mathbf{49}}$
darkturquoise	`\dfrac{\darkturquoise{\partial}f}{\darkturquoise{\partial}x}`	$\dfrac{\darkturquoise{\partial}f}{\darkturquoise{\partial}x}$
onlylight	`\onlylight{\LaTeX}`	$\onlylight{\LaTeX}$
lightanddark	`\lightanddark{\sum}_{n=1}^\infty`	$\lightanddark{\sum}_{n=1}^\infty$
hyphen-color	`\class{hyphen-color}{\sum}_{n=1}^\infty`	$\class{hyphen-color}{\sum}_{n=1}^\infty$

All of the $\LaTeX$ commands can be used in all components that already support $\LaTeX$.

MarkDown text colors

The defined roles can be used in regular MarkDown code, similar to other roles such as `numref` and `code`.

Color	Style	Markdown Code	Result
olive	regular	`{olive}`regular``	regular
hotpink	strong	`{hotpink_strong}`strong``	**emphasis**
onlylight	emphasis	`{onlylight_emphasis}`emphasis``	*emphasis*
lightanddark	strong emphasis	`{lightanddark_strong_emphasis}`strong emphasis``	***strong emphasis***
hyphen-color	regular	`{hyphen-color}`regular``	regular

Colored admonitions

Any existing admonition supporting the `class` option, whether provided by Sphinx or by a Sphinx extension, can be given a different color by adding a CSS/custom named color to the list of classes.

An alternative option is to use an admonition with the name of the CSS/custom named color and add the type of admonition to the list of classes. In that case and for admonitions without the title argumentbut an automatic title (such as `wanring`), a title has to be set explicitly. This alternative approach does take over numbering (if any) of the oringinal admonition type (if any).

Following are some examples with different colors, with the two code options next to each other, followed by the two results.

A special feature is a new class for existing admonitions: `no-title`. This suppresses printing of the title, even if the title is given. For the named color admonitions this happens automatically if no title is given.

General admonition

```
::::{admonition} General admonition with title
:class: olive
Content of general admonition.
::::
```

```
::::{olive} General admonition with title
Content of general admonition.
::::
```

General admonition with title

Content of general admonition.

General admonition with title

Content of general admonition.

Dropwdown admonition

```
::::{admonition} Dropdown admonition with title
:class: hotpink, dropdown
Content of general admonition.
::::
```

```
::::{hotpink} Dropdown admonition with title
:class: dropdown
Content of general admonition.
::::
```

Dropdown admonition with title

Content of general admonition.

Dropdown admonition with title

Content of general admonition.

Common admonitions

```
::::{warning}
:class: darkturquoise
Content of warning.
::::
```

```
::::{darkturquoise} Warning
:class: warning
Content of warning.
::::
```

Warning

Content of warning.

Warning

Content of warning.

Admonitions from Sphinx-Proof

```
::::{prf:defintion}
:class: onlylight
Content of definition.
::::
```

```
::::{onlylight} Definition
:class: definition
Content of definition.
::::
```

Definition 1

Content of definition.

Definition

Content of definition.

Titleless admonitions

```
::::{admonition} This title will not be shown
:class: lightanddark, no-title
Content of admonition.
::::
```

```
::::{lightanddark}
Content of admonition.
::::
```

Content of admonition.

Content of admonition.

Titleless admonitions with dropdown

In case a dropdown admonition is required without a title, only the new CSS/custum named color admonitions can safely be used.

```
::::{lightcoral}
:class: dropdown, warning
Content of admonition.
::::
```

Content of admonition.

Titleless admonitions with title bar

In case an admonition is required without a title but with a title bar, only the new CSS/custum named color admonitions can safely be used. In that case add the class `show-bar`.

```
::::{gold}
:class: show-bar, warning
Content of admonition.
::::
```

Content of admonition.

Contribute

This tool's repository is stored on GitHub. If you'd like to contribute, you can create a fork and open a pull request on the GitHub repository.

The `README.md` of the branch `manual` is also part of the TeachBooks manual as a submodule.

TeachBooks Student-view Features

This chapter includes some TeachBooks features which aim to improve the way students can use TeachBooks. These tools are fully student-side, they don’t influence the book website and the author doesn’t have to setup anything. Nevertheless, the author could point the students to the availability of these tools and use it as part of the didactical approach. These tools could also be useful for other users.

Local Annotator Extension

Contents

A tool for making annotations on websites, with the primary application for use with online interactive textbooks. This extension provides students, readers and anyone with the ability to use an online textbook in a similar way as a paper book: highlight text and make notes in the margins. The extension is developed by the TeachBooks team (info@teachbooks.io)

You can download the extension directly from the Chrome Web Store.

If you want to try it in **developer mode**, follow the step-by-step instructions below.

Features

Current Features

- **Multiple Highlight Colors**: Choose from four different highlight colors (Yellow, Pink, Green, Blue) to organize your highlights.
- **Annotations**: Add annotations with a clean red underline style.
- **Filtering System**:
  - Toggle visibility of annotations
  - Toggle visibility of specific highlight colors
  - Filters persist across page refreshes
- **Export/Import System**:
  - Selectively export annotations and highlights
  - Choose which color highlights to include in export
  - Export data in JSON format
  - Import previously exported data
  - Share annotations and highlights with others through exported files
- **Statistics View**: See a summary of your highlights and annotations for the current page
- **Search Functionality**: Search through your highlights and annotations
- **Overlap Prevention**:
  - Prevents overlapping highlights
  - Prevents overlapping annotations
  - Prevents highlighting annotated text
- **Contextual Toolbars**:

    - Floating toolbar for text selection
    - Annotation toolbar for managing existing annotations
- **Annotation Sidebar**:
    - View all annotations for the current page
    - Edit and delete annotations
    - Animated scrolling to selected annotations
- **Persistent Storage**: All highlights and annotations are saved locally and persist across sessions
- **Extension State Management**: Ability to activate/deactivate the extension
- **Multi-block Selection**: Support for selecting text across multiple blocks/paragraphs
- **Toast Notifications**: User-friendly feedback for actions

Future Features

- **Tags for Annotations**: Allow users to categorize annotations using custom tags
- **Organized Dashboard**: A centralized dashboard to manage annotations across multiple pages
- **Multi-page View**: Navigate and manage annotations across multiple tabs or pages
- **Enhanced Export Options**: Export in multiple formats (PDF, Markdown, CSV)
- **Collaborative Features**: Real-time collaboration on annotations with others

---

Installation

From Chrome Web Store

1. Visit the Chrome Web Store link.
2. Click the **“Add to Chrome”** button.
3. Once installed, the Local Annotator extension icon will appear in your Chrome toolbar.

Developer Mode

1. Clone or download this repository
2. Run `npm install` to install dependencies
3. Run `npm run build` to build the extension
4. Load the extension in Chrome:
    - Open Chrome and navigate to `chrome://extensions/`
    - Enable “Developer mode”
    - Click “Load unpacked” and select the `dist` directory

How to Use

Highlighting Text

1. Select any text on the page
2. Click one of the highlight color buttons in the floating toolbar

3. Toggle highlight visibility using the filter menu in the extension popup

Adding Annotations

1. Select text you want to annotate
2. Click the annotation button in the floating toolbar
3. Add your annotation in the sidebar that appears
4. The annotated text will be marked with a red underline

Managing Visibility

1. Click the extension icon to open the popup
2. Use the “Filters” tab to:
    - Toggle annotation visibility
    - Toggle specific highlight colors
3. Filter settings persist across page refreshes

Exporting Data

1. Click the extension icon
2. Click “Export Data”
3. Select what to include in the export:
    - Choose which highlight colors to export
    - Choose whether to include annotations
4. Click “Export Selected” to download the JSON file

Importing Data

1. Click the extension icon
2. Click “Import Data”
3. Select a previously exported JSON file
4. Your highlights and annotations will be restored

Viewing Statistics

1. Click the extension icon
2. The “Statistics” tab shows:
    - Count of highlights by color
    - Total number of annotations
    - All statistics for the current page

Technical Challenges: The Overlap Problem

One of the most significant challenges during development was handling overlapping highlights and annotations. This is an inherently complex problem due to the nature of DOM manipulation and the various ways text can be selected across different HTML elements.

Attempted Solutions

1. **DOM Tree Traversal**:
   - Attempted to traverse the DOM tree to find and split overlapping ranges
   - Failed because nested highlights created complex DOM structures that were difficult to traverse reliably
   - Led to inconsistent results when multiple highlights overlapped
2. **Range Intersection Detection**:
   - Tried to detect intersections between ranges before applying highlights
   - Worked for simple cases but failed with complex selections spanning multiple DOM elements
   - Couldn't handle cases where selections partially overlapped existing highlights
3. **DOM Fragment Manipulation**:
   - Attempted to extract content, modify it, and reinsert it
   - Led to issues with DOM structure integrity
   - Lost event listeners and broke existing highlights
4. **Layer-based Approach**:
   - Tried to create separate layers for different highlights
   - Failed because it required significant DOM restructuring
   - Caused issues with text selection and copy/paste functionality
5. **Virtual DOM Implementation**:
   - Attempted to maintain a virtual representation of the highlights
   - Synchronization between virtual and actual DOM became extremely complex
   - Performance issues with large numbers of highlights
6. **CSS-only Solution**:
   - Tried using pure CSS for highlighting without DOM manipulation
   - Couldn't achieve the required precision for text selection
   - Failed to handle multi-block selections properly

Current Implementation

Due to these challenges, the current implementation prevents overlapping entirely. When a user tries to create a highlight or annotation that would overlap with existing ones, the operation is blocked and a user-friendly message is displayed. This decision was made because:

1. **DOM Stability**:
   - Each highlight and annotation modifies the DOM structure
   - Overlapping modifications can lead to unpredictable results
   - Maintaining DOM integrity is crucial for proper functionality
2. **User Experience**:
   - Clear feedback when overlap is detected
   - Consistent behavior across different scenarios

    - No risk of corrupting existing highlights or annotations
3. **Future Maintainability**:
    - Clean separation between different highlights and annotations
    - Simpler codebase without complex overlap handling
    - More reliable storage and restoration of highlights

Future Considerations

While overlapping highlights and annotations remain unsupported, potential future solutions might include:

- Implementing a layer-based system using modern web technologies
- Using Web Components for better encapsulation
- Exploring new DOM manipulation techniques as browsers evolve

---

Feedback and Contributions

Feel free to submit feedback or suggest new features by creating an issue in this repository. Your input helps make the Local Annotator better for everyone!

Making comments on the website: Hypothesis

Contents

Hypothesis is a third-party application that allows anyone to make publicly visible comments on a website (once you create a free account). This section explains how to use and install the extension. As this tool requires an account with a third-party, we advise to use our local-annotator with similar capabilities but which store annotations locally. By default Hypothesis stores annotations publicly, which might confuse other students.

> ⚠ **Be careful of changing website URL's!**
>
> Annotations created using the Hypothesis feature are linked to the specific URL of a website, so keep this in mind if you move a book, or change the file name, as you could lose track of your work! Note that while changing the URL may make a page invisible, you can still find the annotations using the *View group activity* feature.
>
> For example, if there is no annotation visible on this page, the URL probably changed!

Set up Hypothesis in your book

Add this to `_config.yml`. Need to create an account.

```
html:
  comments:
    hypothesis: true
```

Using Hypothesis

Once on a hypothesis version of this page, look at the top right corner: you will see three icons, an arrow, an eye and a note-pad. Clicking on the arrow will open the extension and allow you to view the annotations. Create an account and log in if you would like to reply or make your own annotations. You can also highlight text, but this is not publicly visible, it is only for your own reference.

Note that the annotations are only visible for the page that you are currently viewing. You can see all the Annotations on a website by opening the "Public" menu at the top of the dialog pane, selecting "Public" again in the drop-down, then "View group activity." Make sure you type in the URL to search (e.g., `teachbooks.io/manual`).

Student-controller with browser extension

As an alternative to activating this for everyone, you can also advise your students to use the hypothesis browser extension. In this way, students are not distracted by potential unnecessary public annotation and comments by fellow students and

teachers.

TeachBooks Examples

This chapter will showcase some TeachBooks making great use of Jupyter Book and TeachBooks features. They can serve as an inspiring starting point for structuring your own TeachBook or just to get an idea of how other teachers and students are using it. Lastly some example pages are shown which combine theory and interactive questions and coding to different degrees.

Well-Structured Book

Contents

One thing that students love is having all practical information about a course collected and updated on one page. No more scrolling through the introduction lecture on Brightspace while veryfing assessment methods on the online studyguide or finding out about the schedule on MyTimetable. As will be shown in the following section, a Jupyter Book can be used to clarify the logistics of a course on top of providing the course material to the students.

The crossover module Data Science and Artifical Intelligence for Engineers, or in short DSAIE, aims to teach powerful data science tools to 2nd master students from the Civil Engineering and Geosciences faculty. In order to do that they use a combination of practical projects and theory provided to the students through their online interactive book.

Table of contents

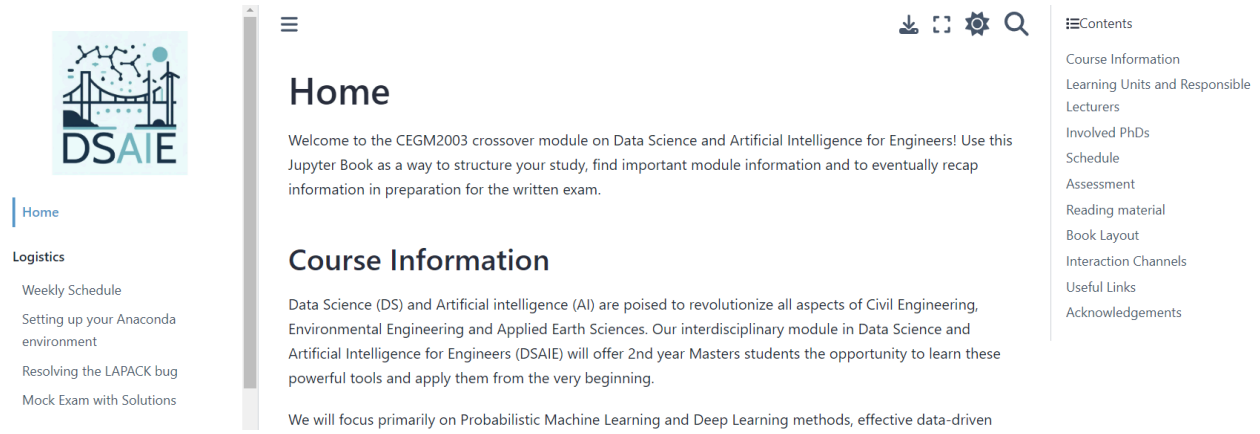


Fig. 116 DSAIE Overview Page

While the left menu shows the first two chapters in the table of contents (Home & Logistics), the right hand side menu shows the sections of the current chapter where an introduction is given on the different units of the course, the responsible lecturers as well as the schedule of

lectures and practicals. Additionally, the book layout is explained to the students which can help clarify how to use the interactive sections!

The anatomy of a jupyter book is explained more in detail on this page. In short, you can create structure in your book by having chapters and subschapters defined in the `.toc` file. In each chapter you can organize the content in sections using the `#` syntax which is visible in the menu on the right.

Tagged content

Since all of this information can get a bit overwhelming it might be useful to tag the content so that students can easily grasp how the course is built up. Clicking on `Assessment` in the contents menu on the right will bring you to the following paragrapgh.

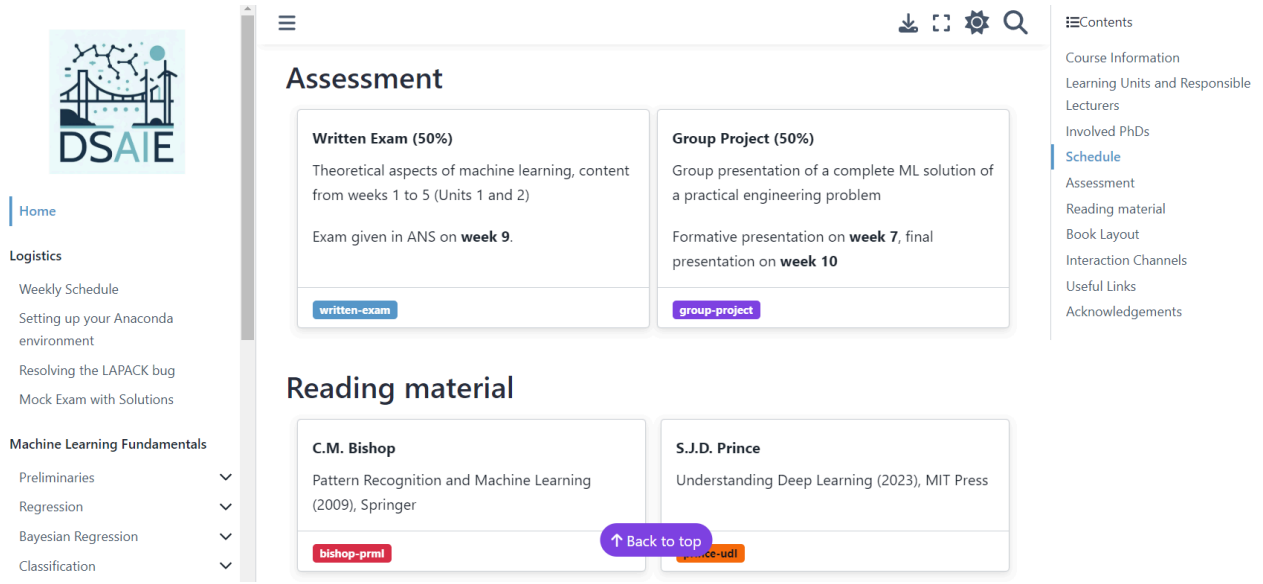


Fig. 117 DSAIE Assessment Methods + Tags

The lecturers have introduced some tags such as `written-exam` and `group-assignment`. Likewise, the reading material is also tagged with different sources so that students know in which textbook they can find more information if interested.

Throughout the book, those tags might be integrated as follows:

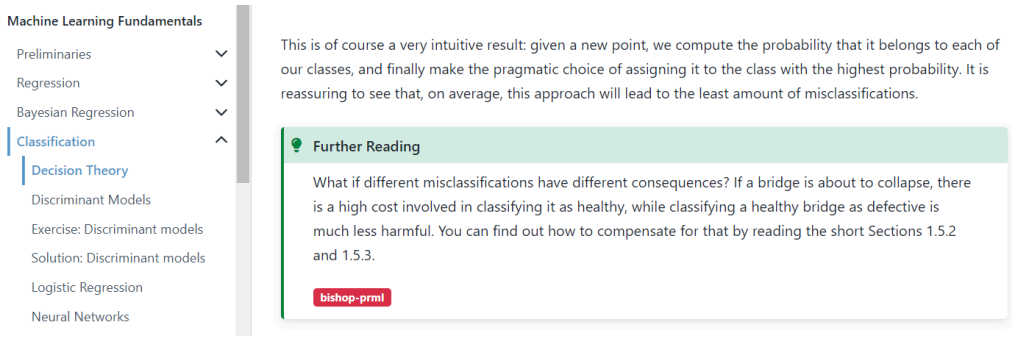


Fig. 118 Additional reading reference

my_new_tag

Make your own!

```
+++
{bdg-primary}`your_new_tag`
```

These badges as they are called are provided by the `spinx-design extension`. Different colours can be found through a bit of research online or here as a starting point.

Live Code (Sphinx Thebe)

Contents

The live-coding option is what makes working with these interactive textbooks extra attractive for courses where coding might be involved or useful to showcase examples. Additionally, it is useful to illustrate the content in a way which the student will recognize during programming assignments.

This page will show some examples where live coding is used in combination with theoretical questions as well as as stand-alone exercises used to round off a theoretcal chapter with a practical application.

Python for Engineers

Python for engineers is a book used to introduce basic programming concepts to students coming from different universities where programming is not a big part of the curriculum like at TU Delft. The students are encouraged to follow the course online before the start of the year to hit the ground running in September. Chapter 7 introduces the students to Matplotlib and teaches them various ways of making a graph.

Live coding is the perfect learning tool as the students can see how the code relates to the visual result of the graph. subsequently, they can adapt the code and observe the changes. Let's have a look at the implementation.

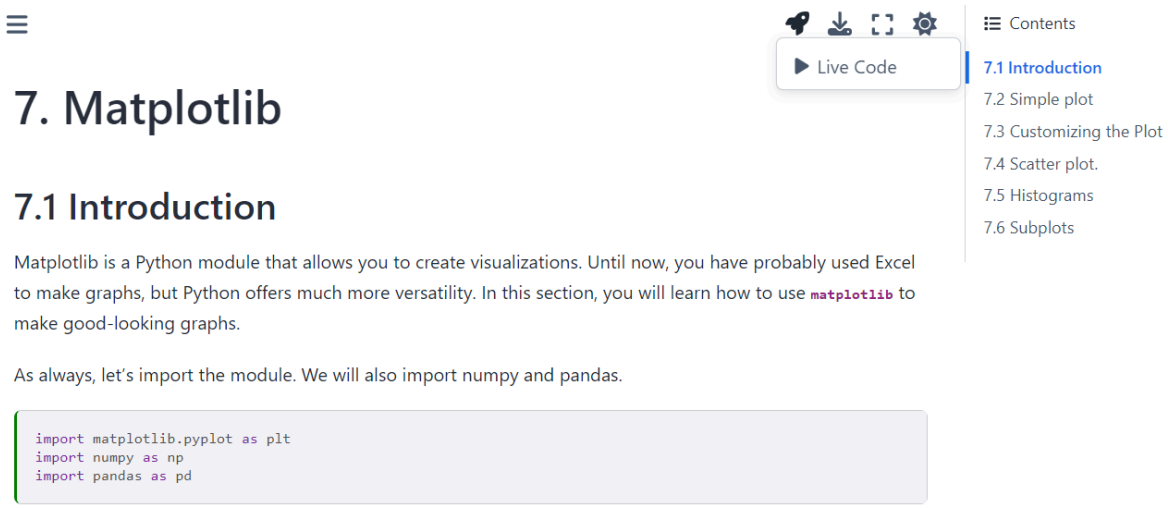


Fig. 119 Rocket Button

To activate the live coding, the rocket button has to be activated like in the figure above. It's always useful to point this out in the beginning of the chapter because it might not be obvious to new users. Here's a note used in the MUDE book.

Note

Interactive Pages — Use Python in your Browser! This online textbook has a number of pages that are set up to be used interactively. You can use the "Live Code" button under the Rocket Ship icon in the top right to activate the interactive features and use Python interactively!

Sometimes the interactivity will involve completing an exercise, wheras on other pages it might simply provide the opportunity to edit the contents of code cells and execute it to explore the page contents interactively. Other pages may provide interactive figures (e.g., widgets).

This feature is supported by TeachBooks.

Once the live coding is activated the code cells run the python code. They even display error messages if there is a mistake in the code.

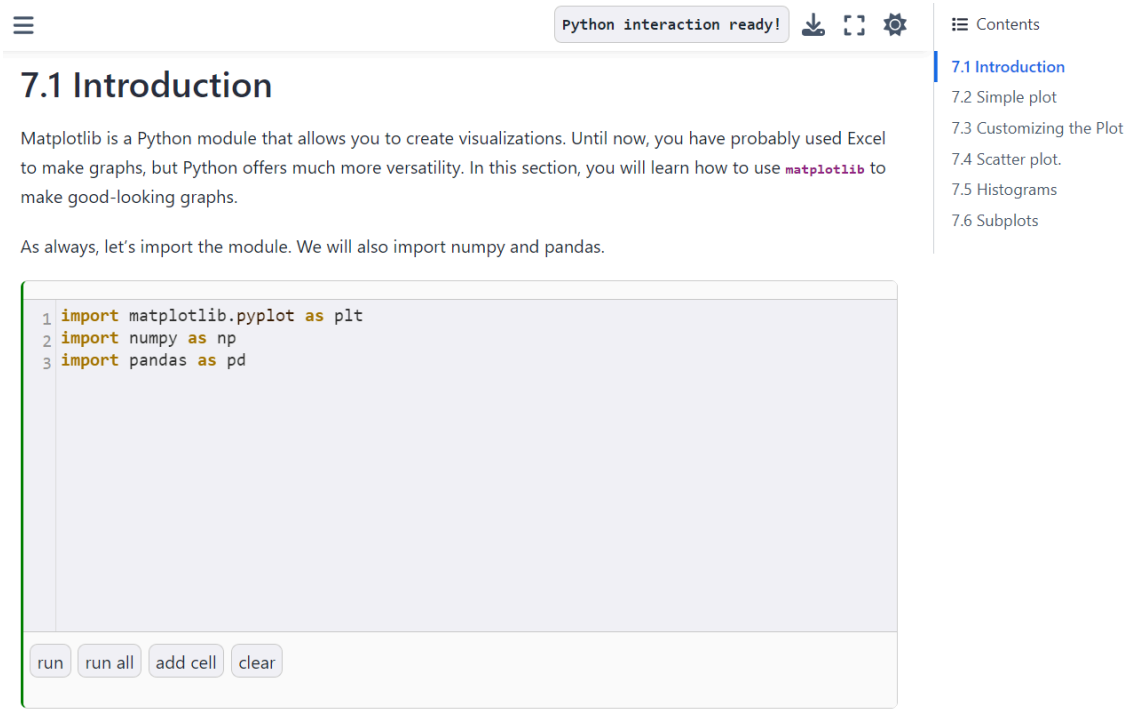


Fig. 120 Python Ready

After introducing some types of plots and breaking down and explaining the lines of code, it is time for the student to have little go themselves. The code is given and they can customize the input.

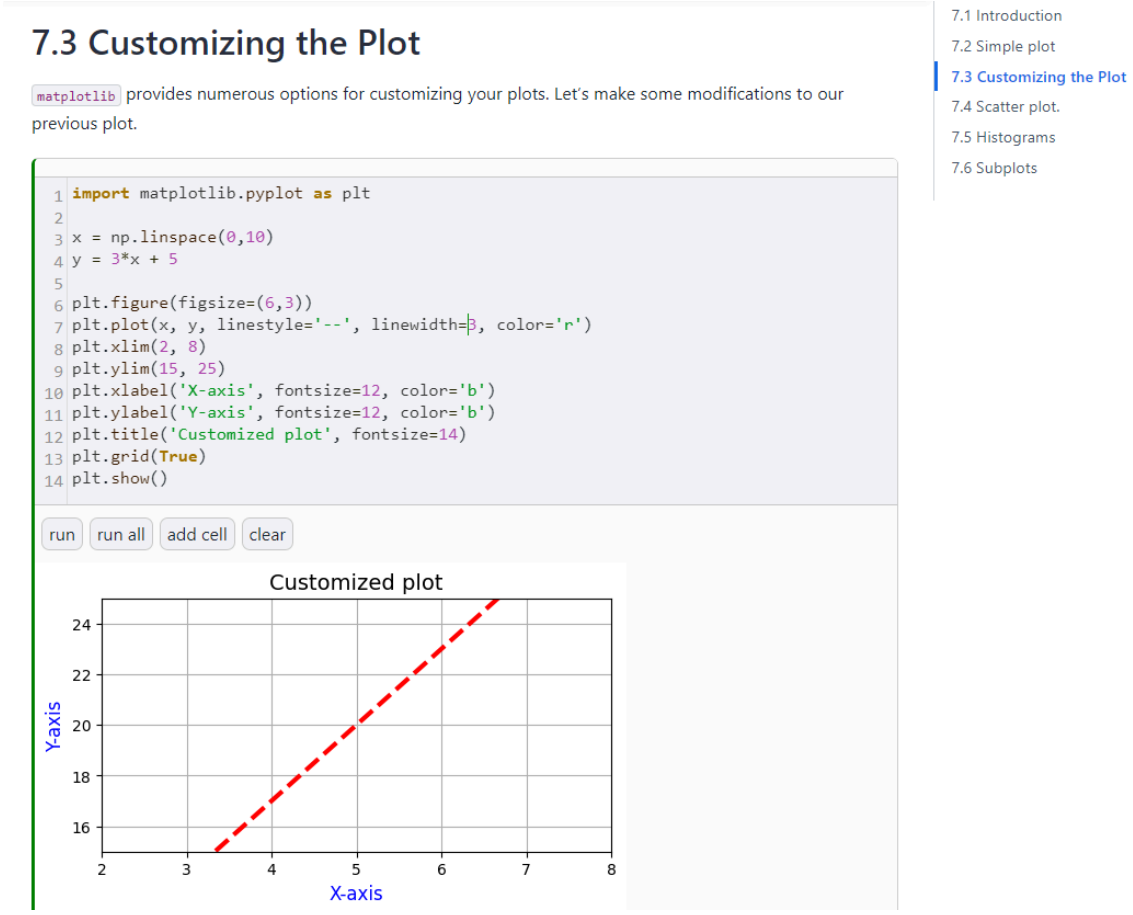


Fig. 121 Graph Customization

This example is rather basic but a similar concept can be applied to more difficult concepts in engineering. The student is getting direct feedback from the code to see how a different input changes the output which reinforces the theory.

MUDE - Signal Processing

In MUDE, the concept of Signal Processing is introduced. On top of the theory, a working example is used to test the understanding of the students. When talking about Signal Processing, the Fourier Series is a foundational block that has to be well understood. The fourier series is used to describe basically any periodic function/signal as a sum of multiple cosines and sines with different frequencies. This is illustrated with the concept of the square wave.

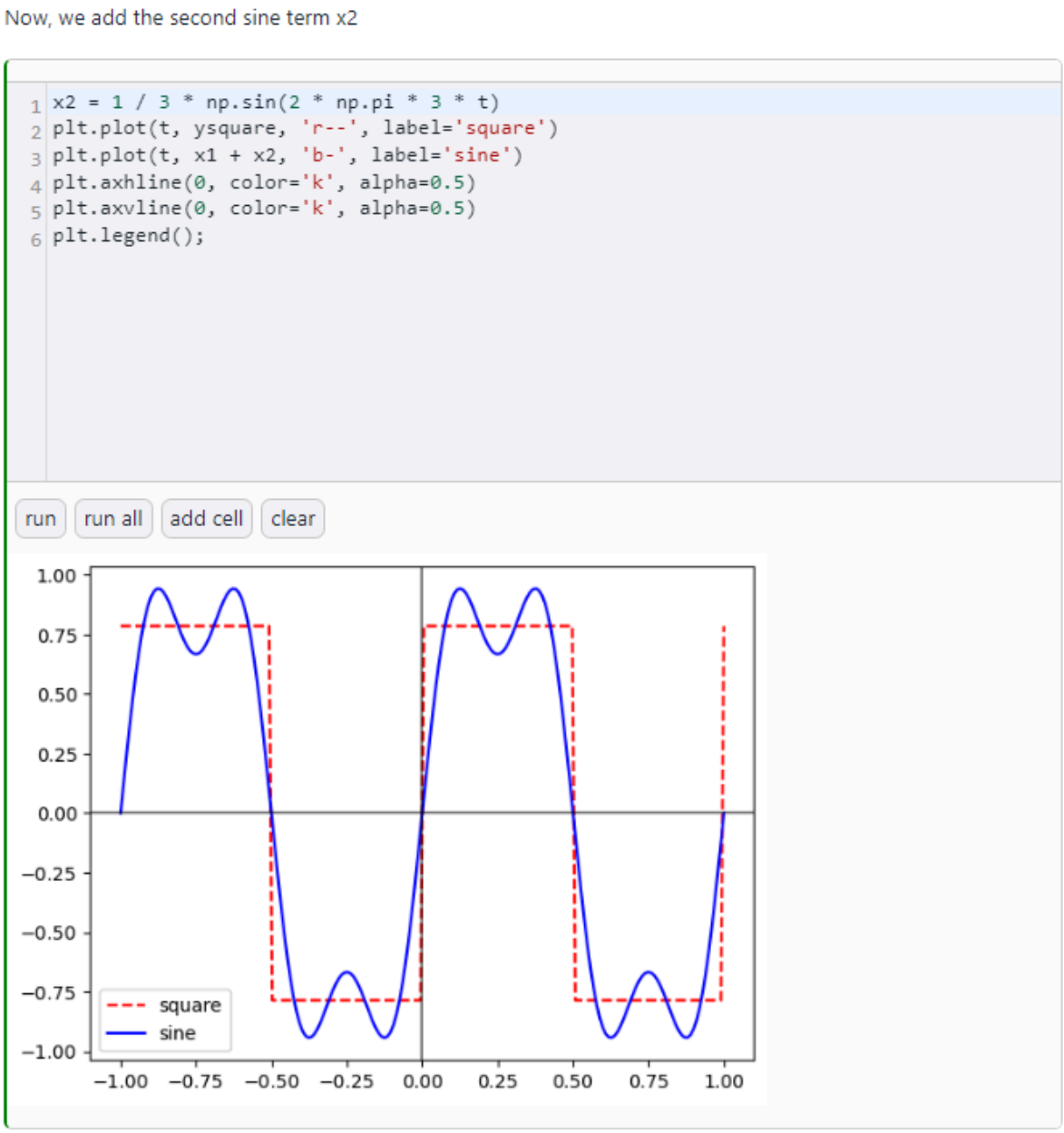


Fig. 122 Superposition of two signals

By adding more signals themselves, the students will see that the fourier series approximates the square wave. They might even see which frequencies work better than others.

Finally lets look at a more computationally heavy example.

MUDE - Machine Learning

In the chapter on the k-nearest neighbour, live-coding is activated by default in order to run all the coding which supports the theory. For more complex topics, it is useful to keep the actual code cells hidden and only display the graphs with interactive sliders in order to focus the learning objective more on the content and less on the coding.

The k-nearest neighbour method predicts the value *f(x)* for a given *x* by taking the average value of its neigbhours and assigning it to *f(x)*. The amount of neighbours considered is governed by the variable *k*.

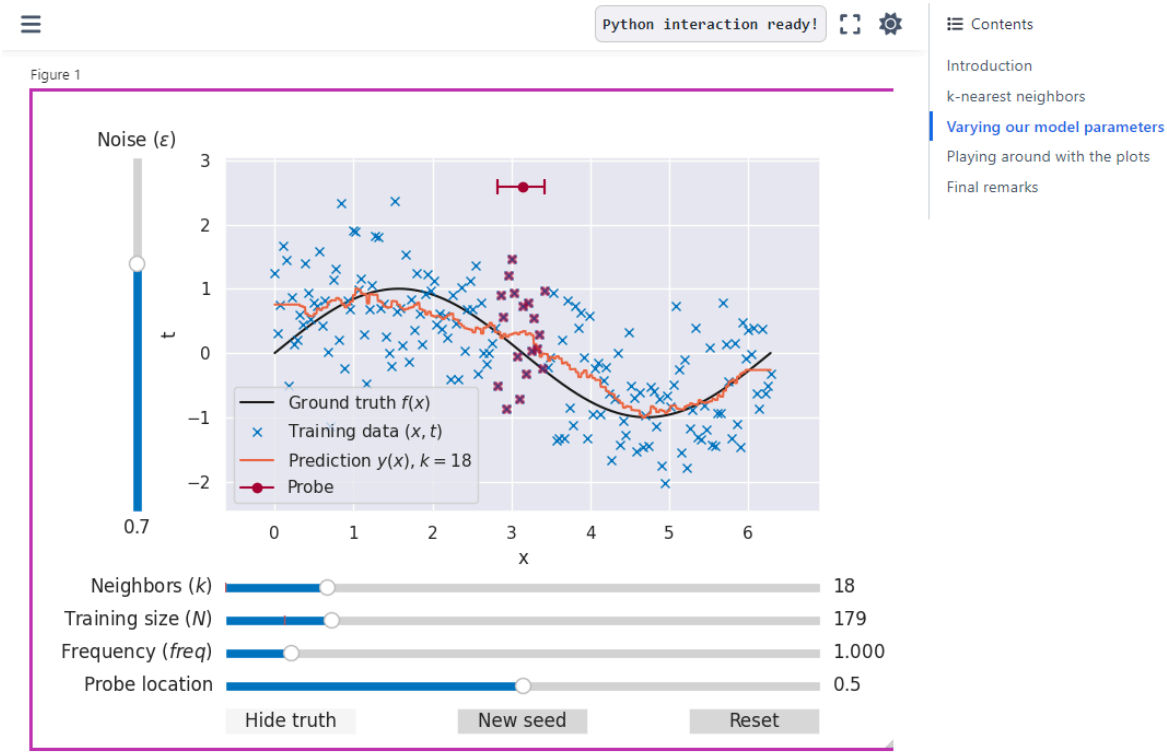


Fig. 123 Interactive Graph

Multiple sliders modify the predicted graph (in orange) by taking into account a different amount of neighbours, training size, frequency of the signal and amount of noise. The student is able to verify the claims made by the theory throught the interactive graph.

Programming assignment with checks

Contents

This page shows how you can recreate the classical Jupyter Notebook assignments that are given in many courses to a Jupyter Book format with live coding. A widget has been implemented to check the answers, of which the data is included in a separate python file. The page automatically loads and a custom download page has been added

> **Note**
>
> Keep in mind that Jupyter Book sections might not be the best environment for programming assignments you have in mind for your course due to the limitation posed by jupyter book. In fact, in jupyter book, python runs in the browser of the students' laptop. Nothing needs to be installed which can be an advantage but also a disadvantage depending on the learning goals of your course. TeachBooks suggests using it's live coding feature to support the theory in textbooks and not as a replacement for Jupyter Notebook.

Gumbel Distribution Exercise

Imagine you are concerned with the concentration of an airborne contaminant, X, measured in ppm. A previous benchmark study estimates that there is a 10% and 1% chance, respectively, of exceeding 4 and 10 ppm, respectively. You have been asked by the regulatory agency to estimate the contaminant concentration with 0.1% probability of being exceeded. Prior studies suggest that a Gumbel distribution can be used to model contaminant concentration, given by the CDF:

$$F_X(x) = \exp\left(-\exp\left(-\frac{x-\mu}{\beta}\right)\right)$$

Using the cell blocks below as a guide (and also to check your analysis): Task 1) find the parameters of a Gumbel distribution that matches the information provided above, then, Tasks 2-3) use it to estimate the concentration with exceedance probability 0.1%.

To complete this assignment, you can use `numpy`, `matplotlib` and from the `math` library, `log` and `e` (these are imported for you when you inialize the notebook).

Task 0: fill in the appropriate fitting points:

```
x_1 = _
p_1 = _
x_2 = _
p_2 = _
```

run | run all | add cell | clear

Task 1: derive the distribution parameters:

```python
def gumbel_2_points(x_1, p_1, x_2, p_2):
    """Compute Gumbel distribution parameters from two points of the
        Arguments:
            x_1 (float): point one
            p_1 (float): cumulative probability for point 1
            x_2 (float): point two
            p_2 (float): cumulative probability for point 2
    """

    # YOUR CODE GOES HERE #
    beta = _
    mu = _
    ########################

    return mu, beta
```

run | run all | add cell | clear

The cells below will print your parameter values and create a plot to help confirm you have the right implementation.

```python
mu, beta = gumbel_2_points(x_1, p_1, x_2, p_2)

print(f"Your mu: {mu:0.5f}\nYour beta: {beta:0.5f}")

gumbel_distribution = lambda x: 1 - e**(-e**(-(x - mu)/beta))

plot_distribution(mu, beta, x_1, p_1, x_2, p_2)
```

run | run all | add cell | clear

Task 2: write a function to solve for the random variable value (it will be tested for you with the Check answer button, along with the distribution parameters).

```python
def find_x_with_probability_p(p):
    """ Compute point in the gumbel distribution for which the CDF is
        Use the variables mu and beta defined above!
        Hint: they have been defined globally, so you don't need to
                include them as arguments.
    """
    # YOUR CODE GOES HERE #
    x = _
    ########################

    return x
```

run | run all | add cell | clear

Task 3: use the function `find_x_with_probability_p` to evaluate the random variable value with exceedance probabiltiy of 0.001.

```python
# YOUR CODE GOES HERE #
x = _
########################

print(f"The value of x with probability of exceedance 0.001 os {x:0.5f
```

run | run all | add cell | clear

Check Answer(s)

Combining theory & interactive quizzes

Contents

This page shows you how to combine theory with questions so there is no coding involved.

Truss structures

Truss structures are structures assembled of straight elements connected in nodes. These elements partly create triangles, which are of importance for its deformability. Of course, the structures you'll design shouldn't deform, so it's crucial to understand this topic.

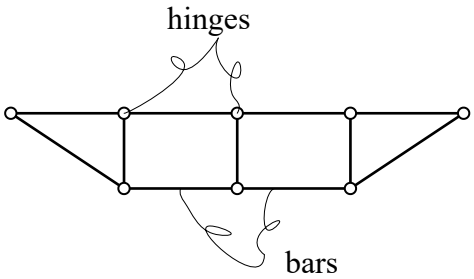
Examples truss structures

Examples of these structures are commonly found in rail bridges like the one below:

Furthermore, these structures are commonly found in building as well, both invisible and visible:

Modelling truss structures

Truss structures are modelled as rigid bars (so elements which cannot deform) connected by hinges (so elements can rotate with respect to one each other). In our model, hinges are indicate with a circle, and bars with a line. For example, the structure you've seen in the second example (with two diagonal bars removed) is modelled as follows:



Now that you've been introduced to truss structures, answer the following question:

Test Yourself

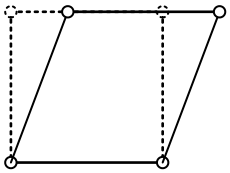
How many hinges and bars does the structure contain?

_____ hinges

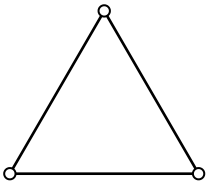
_____ bars

Mechanisms

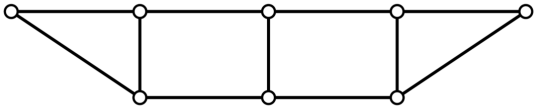
A mechanism is a structure which can deform as a whole, while the individual elements don't deform. This deformation doesn't have to be large. An example of a mechanism is a rectangular truss structure, it can deform as shown below:



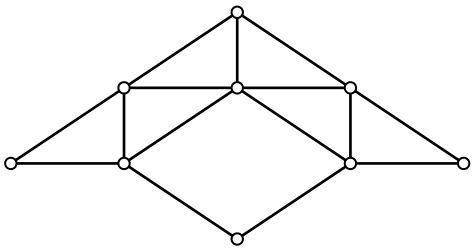
A triangle cannot deform:



For structures which consists of partly triangles, partly quadrilaterals, it is not directly clear whether the structure can deform. To identify whether the truss structure is a mechanism, you can try to move each node with respect to the nodes with which it is connected:



An example of a truss structure of partly triangles, partly quadrilaterals, which is not a mechanism is the following:



Now that you've been introduced to mechanisms, answer the following questions:

Test Yourself

Which sentence is true?

- [ ] A truss stucture can be a mechanism
- [ ] A truss structure is always a mechanism
- [ ] A mechanism can be a truss structure

Check

Test Yourself

Which sentence(s) is/are true?

- [ ] A truss structure which contains both triangles and quadrilaterals is always a mechanism
- [ ] A truss structure which is a mechanism cannot contain any triangles
- [ ] A truss tructure which contains both triangles and quadrilaterals might be a mechanism.
- [ ] A truss structure which is a mechanism should contain only triangles

Theory, interactive quizzes & live code

Contents

This page shows an example of how to combine theory, quizzes and code on one page. With headings and boxes, the interactive parts can be indicated (for now only if part of one cell). Automatic execution has been activated and a custom ipynb.file has replaced the download button.

Weighted least-squares estimation

In ordinary least-squares estimation, we assume that all observations are equally important. In many cases this is not realistic, as observations may be obtained by different measurement systems, or under different circumstances. We want our methodology for least-squares estimation to be able to take this into account.

We achieve this goal by introducing a weight matrix in the minimization problem:

$$\min_{\mathrm{x}} (\mathrm{y} - \mathrm{Ax})^{\mathrm{T}} \mathrm{W} (\mathrm{y} - \mathrm{Ax})$$

In the unweighted least-squares approach, we arrive at the normal equation by pre-multiplying both sides of $\mathrm{y} = \mathrm{Ax}$ with the transpose of the design matrix A^{T}:

$$\mathrm{y} = \mathrm{Ax} \;\rightarrow\; \mathrm{A}^{\mathrm{T}}\,\mathrm{y} = \mathrm{A}^{\mathrm{T}}\,\mathrm{Ax}$$

In the weighted least-squares approach, we now need to add weight matrix W to this pre-multiplication factor, i.e., $\mathrm{A}^{\mathrm{T}}\mathrm{W}$, to obtain the normal equation:

$$\mathrm{y} = \mathrm{Ax} \;\rightarrow\; \mathrm{A}^{\mathrm{T}}\mathrm{W}\,\mathrm{y} = \mathrm{A}^{\mathrm{TW}}\,\mathrm{Ax}$$

The normal matrix is now defined as $\mathrm{N} = \mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A}$. From this, assuming that the normal matrix N is invertible (non-singular) we find the weighted least-squares estimate $\hat{\mathrm{x}}$,

$$\begin{aligned}\hat{\mathrm{x}} &= (\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{y} \\ &= \arg\min_{\mathrm{x}} (\mathrm{y} - \mathrm{A}\mathrm{x})^{\mathrm{T}}\mathrm{W}(\mathrm{y} - \mathrm{A}\mathrm{x})\end{aligned}$$

We also find the derived estimate $\hat{\mathrm{y}}$ and $\hat{\epsilon}$:

$$\hat{\mathrm{y}} = \mathrm{A}\hat{\mathrm{x}} = \mathrm{A}(\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{y}$$

$$\hat{\epsilon} = \mathrm{y} - \hat{\mathrm{y}} = \mathrm{y} - \mathrm{A}\hat{\mathrm{x}} = \mathrm{y} - \mathrm{A}(\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{y} = (\mathrm{I} - \mathrm{A}(\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W})\mathrm{y}$$

Quiz question

You and your fellow students are asked to measure the width of a canal. You all decide to take a different approach. Alice is asked to come up with an estimate of the canal width using all m observations. Based on her own 'ranking', Alice decides to give different weights to the different observations. Which of the following is the equation that Alice will use to estimate the canal width? The individual weights are called w_i, $(i = 1, \ldots, m)$

- $\hat{x} = \dfrac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} y_i$
- $\hat{x} = \sum_{i=1}^{m} \dfrac{y_i}{w_i}$
- $\hat{x} = \dfrac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} (w_i y_i)$
- $\hat{x} = \dfrac{1}{m} \sum_{i=1}^{m} y_i$

Check

Video

Discussion on the weight matrix

The weight matrix W expresses the (relative) weights between the observations. It is always a square matrix. The size of the weight matrix depends on the number of observations, m. The size of the weight matrix is $m \times m$.

If it is a unit matrix ($\mathrm{W} = \mathrm{I}$), this implies that all observations have equal weight. Note that in this case the equations are equal to the ordinary least-squares solution.

If it is a diagonal matrix, with different values on the diagonal, this implies that entries with a higher value correspond to the observations which are considered to be of more importance. If the weight matrix has non-zero elements on the off-diagonal positions, this implies that (some) observations are correlated.

Weighted least-squares estimator: properties

Until now, we have looked at the weighted least-squares solution of a single *realization* of the observations, where generally we assume that it is a realization of the *random* observable vector Y, since measurements are affected by random errors. As such it follows the the weighted least-squares *estimator* is given by:

$$\hat{X} = (\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}Y$$

This estimator has two important properties: it is *linear* and *unbiased*.

The linearity property is due to the fact that $\hat{X}$ is a linear function of the observables Y.

The unbiased property means that the expectation of $\hat{X}$ is equal to the true (but unknown) $\mathbf{x}$. This can be shown as follows:

$$\mathbb{E}(\hat{X}) = (\mathbf{A}^{\mathrm{T}}\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{W}\mathbb{E}(Y) = (\mathbf{A}^{\mathrm{T}}\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{W}\mathbf{A}\mathbf{x} = \mathbf{x}$$

This a very desirable property. It applies that if we would repeat the measurements many times to obtain a new estimate, the *average of the estimated* values would be equal to the truy values.

Exercise

You have a time series of 8 measurements and fit a model assuming a linear trend (constant velocity).

Times of observations, observed values and number of observations are given as the variables `t`, `y` and `m`, as well as `numpy` and `matplotlib.pyplot` abbreviated as `np` and `plt`

Fill in the correct $\mathbf{A}$-matrix.

```
# design matrix
A = np.column_stack((np.ones(m), t))
```

run | run all | add cell | clear

Check answer

Define the weight matrix for $\mathrm{W} = I_m$

```
# Weight matrix for case 1
W_1 = ?
```

run | run all | add cell | clear

Check answer

Define the weight matrix with the weight of first 3 observations is five times as large as the rest

```
# Weight matrix for case 2
W_2 = ?
```

run | run all | add cell | clear

Check answer

Define the weight matrix with the weight of 4th and 5th observation is five times as large as the rest

```
# Weight matrix for case 3
W_3 = ?
```

run | run all | add cell | clear

Coding theory, interactive quizzes & live code

Contents

This page shows an example on how to combine the introduction of a certain coding implementation with interactive questions. Compared to the previous template, the learning objective is to learn a new skill in programming using the live coding feature whereas before programming was used to illustrate a theoretic concept unrelated to coding (for example statistics).

Example unconstrained optimization

In this chapter, we'll cover how to apply `scipy.optimize.minimize` to unconstrained optimization problems. As a reminder, unconstrained optimization considers:

$$\min_x f(x) \tag{4}$$

with x the design variable of length n and f the objective function.

Method

In this course, we're making use of the function `scipy.optimize.minimize`. The documentation of this function is available here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html. In this course we'll cover only the relevant parts.

For unconstrained optimization we need to run at least `scipy.optimize.minimize(fun, x0, ...)` with:

- `fun`, the objective function $f(x)$ to be minimized. `fun` is a callable. The `scipy.optmize.minimize` function takes care of defining and inputing our design variable

x.

- `x0`, the initial guess for our design variable x. It needs to be a `ndarray` with length n

The function `scipy.optimize.minimize` outputs an object `scipy.optimize.OptimizeResult`. with:

- `scipy.optimize.OptimizeResult.x` the optimized solution of the design variable `x`. It is a `ndarray` with length n
- `scipy.optimize.OptimizeResult.success`, a indication whether or not the optimizer was executed succesfully. It is a `bool`, indicating `True` or `False`
- `scipy.optimize.OptimizeResult.message`, a message describing the cause of termination of the optimizatino algorithm. It is a `str`.
- `scipy.optimize.OptimizeResult.fun`, the values of the optimized objective function f. It is a `int` or `float`
- `scipy.optimize.OptimizeResult.nit`, the number of iteration performed by the optimizer. It is a `int`

How many possible input does scipy.optimize.minimize have?

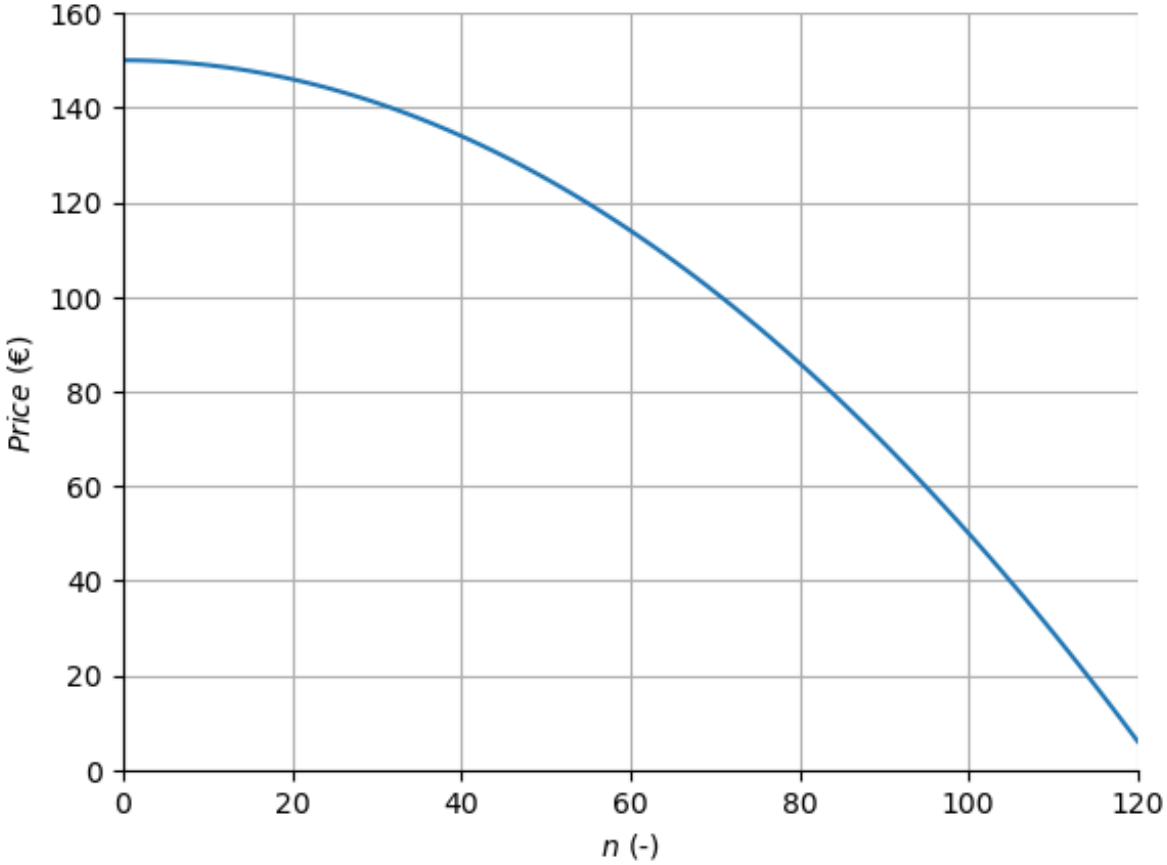
Check

Example 1

Problem

It is desired to determine the number of bathymetry maps n of a local area that should be produced to maximize the profit of a company. The total cost of production and distribution is €75 per unit n. The revenues are proportional to the number of units multiplied by its price: $Revenues = n \cdot Price$

The demand depends on the price ($Price = 150 - 0.01n^2$), as shown in the graph:



The profit can be estimated as the revenues minus the total costs.

Model

The function for the profit can be found by combining the relations in the problem statement. However, this is the profit which should be maximized. To turn this into a minimization problem, the profit can be multiplied with -1. The final model of this problem results in:

$$\min_{n} \left(75n - \left(150 - 0.01n^2\right)n\right) \tag{5}$$

This problem is

- ○ Linear
- ○ Nonlinear

Check

Method

This model is described using `scipy.optimize.minimize`.

Importing libraries

```
1 import scipy as sp
2 import numpy as np
3 import matplotlib.pylab as plt
```

run | run all | add cell | clear

Defining the variables

There are very few variables in this problem. In fact, the only variable we have to specify is the initial guess for the optimization algorithm. The objective function will be treated later. The length of n doesn't have to be specified.

```
1 n0 = 20
```

run | run all | add cell | clear

The length of the design variable n doesn't have to be specified. How do you think the optimization method can proceed?

- ○ It doesn't need to know the length of n
- ○ It uses the length of the initial guess n_0 as a length for n
- ○ It uses the output of the objective function f as a length for n
- ○ It uses the input of the objective function f as a length for n

Check

Defining the objective function

In the objective function, the formula given in the model description can be inserted. Or, each individual step can be calculated on a seperate line. Again, note that the profit is multiplied with -1 to maximize the profit in the minimization formulation. This results in:

```python
def negprofit(n):
    price = 150 - 0.01 * n**2
    revenues = price * n
    totalcost = 75 * n
    profit = revenues - totalcost
    return -profit
```

run | run all | add cell | clear

Solving the problem

Now, the problem can be solved. The result is stored in the variables `result` which is printed.

```
result = sp.optimize.minimize(negprofit,n0)
print(result)
```

run | run all | add cell | clear

```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: -2499.9999999998727
        x: [ 5.000e+01]
      nit: 8
      jac: [ 0.000e+00]
hess_inv: [[ 3.503e-01]]
     nfev: 22
     njev: 11
```

Which solver(s) might be used by `scipy.optimize.minimize`?

- [ ] Nelder-Mead
- [ ] Powell
- [ ] CG
- [ ] BFGS
- [ ] Newton-CG
- [ ] L-BFGS-B
- [ ] TNC
- [ ] COBYLA
- [ ] SLSQP
- [ ] trust-constr
- [ ] dogleg
- [ ] trust-ncg
- [ ] trust-exact
- [ ] trust-krylov

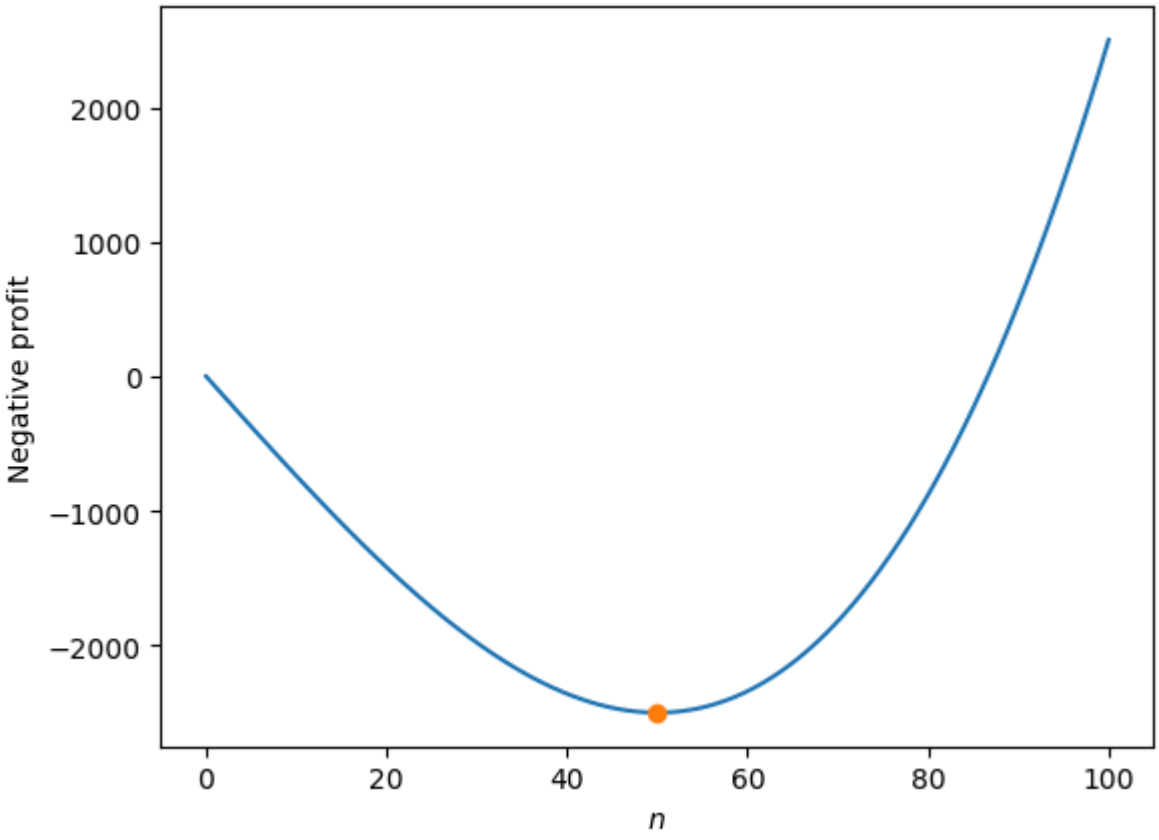
Check

Postprocessing

As this case is only one-dimensional and the potential range of values is limited, we can easily check this answer by an exhaustive search, evaluating all possible values for n. The plot below shows the $\mathrm{negative\ profit}$ for $0 < n < 100$. It shows a clear minimum which coincides with the minimum found by `scipy.optimize.minimize`

```
n_range = np.linspace(0,100,100)
negprofit_result = negprofit(n_range)
plt.plot(n_range,negprofit_result)
plt.plot(result.x,result.fun,'o');
plt.xlabel('$n$')
plt.ylabel('Negative profit');
```

run | run all | add cell | clear



Was it necessary to plot the all possible options to check the optimality of the result of optimization? Think of why it is or isn't?

- ○ Yes
- ○ No

Check

Exercise

Adapt the code to answer the following questions.

What is the optimum number of bathymetry maps if the costs of production are €100 ? Round you answer to the nearest integer

Check

What is the output of the optimisation if the costs of production exceeds €150?

Turn

Card 1 of 2

Parametric questions

Contents

On this page an example is given for parametric questions, in which an equation is checked on equivalence with the correct answer.

Exercises on Taylor expansion

This page shows some exercises on calculation Taylor expansion. If you reload this page, you'll get new values.

Exercise 1

Calculate the taylor series expension of:

$3x^2$

around:

$x = 4$

Fill in your answer and run the cell before clicking 'Check answer'.

```
1 eq1_answer =
```

run | run all | add cell | clear

Check answer

Exercise 2

Calculate the taylor series expension of:

$3\tan(x)$

around:

$x = -2\pi$

discard any $O(x^3)$ terms.

Fill in your answer and run the cell before clicking 'Check answer'. Furthermore, use `pi` for π :

```
1 eq2_answer =
```

run | run all | add cell | clear

Check answer

Exercise 3

Calculate the taylor series expension of:

$$\frac{5}{1-x}$$

around:

$$x = -1$$

discard any $O(x^3)$ terms.

Fill in your answer and run the cell before clicking 'Check answer':

```
1 eq3_answer =
```

run | run all | add cell | clear

Check answer

Figure syntax maker

One might quickely forget the code for making figures, here is a useful tool to make your figures, just specify all information and the code is produced for you:

Label:

Image Location + name + extension:

Caption:

Select type:

○Figure

○Table

Submit

Stored Array:

- **Figuur code:**
  ```{figure} figures/usertype1.jpg
  ---
  name: usertype1
  ---

  ```
  Figuur referentie:
  {numref}`{number} <usertype1>` Remove
- **Figuur code:**
  ```{figure} ./structure.svg
  ---
  name:
  ---

  ```
  Figuur referentie:
  {numref}`{number} <>` Remove
- **Figuur code:**
  ```{figure} https://files.mude.citg.tudelft.nl/
  ---
  name: figure_label
  ---
  caption
  ```
  Figuur referentie:
  {numref}`{number} <figure_label>` Remove
- **Figuur code:**
  ```{figure} https://files.mude.citg.tudelft.nl/
  ---
  ```

name: figure_label

---

caption

```

**Figuur referentie:**

{numref}`{number} <figure_label>` Remove
```

Scripts for automating workflow

This chapter contains a few scripts which can be useful in the book-editing process:

- LaTeX to markdown conversion to convert a LaTeX file into jupyter-book compatible markdown.
- Snippitall to insert the same code snippet at a certain point in each md file in a specific folder and all subfolder.
- Snippitonce to insert the same code snippet at a certain point in each md file only in a specific folder.
- Extensionchecker to identify the kind of files present in subfolder
- Filedownloader to return the downloadlink for markdown for all files with a specified extension
- Figreturner to return the markdown code for a figure in a specific extension
- Figshrinker to compress JPEG images.

Convert LaTeX to Markdown

A script is to convert LaTeX files to Markdown for use in Jupyter Books (Idema (Delft University of Technology), 2024).

This script has been used successfully on at least one book in the TU Delft OPEN library: Introduction to particle and continuum mechanics, by Timon Idema. The author is able to convert nearly everything in this book automatically, with only a few manual adjustments needed (e.g., a special character and a figure caption).

Depending on the number of custom LaTeX commands used in your book, you may need to adjust the script to handle them.

Additional information about the script, along with the source code itself which is distributed under a BSD 3-clause license, is available at the following link on TU Delft GitLab: gitlab.tudelft.nl/opentextbooks/latex-to-markdown-conversion.

Move images to separate hosting

Storing big binary files like images in your git repository and book is not good practice: diffs on binary files don't make sense and blow up the size of your repository. Furthermore, the size of your book website blows up too. Therefore, you can consider hosting your binary files on a different server, as explained here in an example for the MUDE-book. To take all the images of your existing book and update the relative links, a Python script has been developed. This script processes Markdown (`.md`) and Jupyter Notebook (`.ipynb`) files listed in a `_toc.yml` file. It identifies local image references (including MyST `figure` syntax) in these files, copies the images to a specified folder (`new_images`), and replaces the local image paths with a public URL (e.g., `https://files.mude.citg.tudelft.nl/<image_name>`).

Snippit

Contents

> This page reuses BSD 3-Clause License content from TeachBooks (2024). ❝Find out more here.

Snippitall

This code inserts a code snippet (defined in code line 6) on a specified place for each md file in the directory of your choosing, including all subfolders.

```
# Snippetall: code for adding code snippet in each md file in a specified directory inclu
import os

# Specify the path to your base folder and line where snippet should be added
directory = 'book/demos'
include_at_line = 1

def insert_code_in_md_files(directory):
    # Define the code snippet to insert
    code_snippet = """\n<div style="clear: both;">\n\n```{figure} ../../figures/open.png\

    # Walk through all directories and subdirectories
    for dirpath, dirnames, filenames in os.walk(directory):
        for filename in filenames:
            if filename.endswith(".md"):  # Check if the file is a Markdown file
                file_path = os.path.join(dirpath, filename)

                # Read the existing content of the file
                with open(file_path, 'r', encoding='utf-8') as file:
                    lines = file.readlines()

                # Insert the code snippet at the specified line
                include_at_line = 1
                if len(lines) > include_at_line:
                    lines.insert(include_at_line, code_snippet)
                else:  # If the file has less than two lines, append the snippet
                    lines.append(code_snippet)

                # Write the modified content back to the file
                with open(file_path, 'w', encoding='utf-8') as file:
                    file.writelines(lines)
                print(f"Updated {file_path}")


insert_code_in_md_files(directory)
```

Snippitonce

This code inserts a code snippet (defined in code line 6) on a specified place for each md file in the directory of your choosing.

```python
# Snippitonce: code for adding code snippet in specific folder
import os

# Specify the path to your folder
specific_folder = 'book/pedagogy'
include_at_line = 1

def insert_code_in_md_files(specific_folder):
    # Define the code snippet to insert
    code_snippet = """\n<div style="clear: both;">\n\n```{figure} ../figures/confirmed.pn

    # Iterate over all files in the specified folder
    for filename in os.listdir(specific_folder):
        if filename.endswith(".md"):  # Check if the file is a Markdown file
            file_path = os.path.join(specific_folder, filename)

            # Read the existing content of the file
            with open(file_path, 'r', encoding='utf-8') as file:
                lines = file.readlines()

            # Insert the code snippet at the second line
            if len(lines) > include_at_line:
                lines.insert(include_at_line, code_snippet)
            else:  # If the file has less than two lines, append the snippet
                lines.append(code_snippet)

            # Write the modified content back to the file
            with open(file_path, 'w', encoding='utf-8') as file:
                file.writelines(lines)
            print(f"Updated {filename}")

insert_code_in_md_files(specific_folder)
```

Extensionchecker, filedownloader and figreturner

Contents

This code walks through a specified folder and all subfolders to identify the kind of files that are present. It returns the extensions.

```
# Extensionchecker: A script that checks what kind of files are in a folder and its s
import os

directory = '..'

def find_file_extensions(directory):
    extensions = set()  # Used to exclude duplicates
    # Iterate over all files in the specified folder
    for root, dirs, files in os.walk(directory):
        for file in files:
            _, ext = os.path.splitext(file)  # Splits the filename to extract the extensi
            if ext:  # Check whether the extension already exists
                extensions.add(ext.lower())  # Adds the extension and uses lowercase
    return print(extensions)

find_file_extensions(directory)
```

Code searches for specified extensions and returns them as a downloadlink for markdown

```python
# Filedownloader: A script that searches for specified extensions and returns them as a d

import os
from urllib.parse import quote

directory = '..'

def find_files(directory):
    markdown_links = []
    # Loop door de directory en alle subdirectories
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith((".pdf", ".jpg", ".docx", )):
                # Creëer het volledige pad naar het bestand
                full_path = os.path.join(root, file)
                # Converteer het pad naar een relatief pad (verwijder de opgegeven root d
                relative_path = os.path.relpath(full_path, directory)
                # Encode het pad voor gebruik in URL
                url_encoded_path = quote(relative_path)
                # Formatteer het pad en de bestandsnaam als een Markdown link


                markdown_link = f"[{file}]{url_encoded_path})"
                markdown_links.append(markdown_link)
    return markdown_links

# Stel de root directory in waar je wilt beginnen met zoeken naar PDFs
links = find_files(directory)

# Print alle gevonden Markdown links
for link in links:
    print(link)
```

A script that searches in a specified folder and all subfolders for figure files (jpg) and returns the code for figures in markdown.

```python
# Return figure for md
# Note that you should replace %5C with / and %20 with space

import os
from urllib.parse import quote

directory = '..'

def find_jpg_files(directory):
    markdown_links = []
    # Loop door de directory en alle subdirectories
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith(".jpg"):
                # Creëer het volledige pad naar het bestand
                full_path = os.path.join(root, file)
                # Converteer het pad naar een relatief pad (verwijder de opgegeven root d
                relative_path = os.path.relpath(full_path, directory)
                # Encode the path for use in URL, preserving slashes and spaces
                url_encoded_path = quote(relative_path, safe='/ ')
                # Formatteer het pad en de bestandsnaam als een Markdown link


                markdown_link = f"``` {directory}/{url_encoded_path} \n--- \nwidth: 50% \
                markdown_links.append(markdown_link)
    return markdown_links

# Stel de root directory in waar je wilt beginnen met zoeken naar PDFs
pdf_links = find_jpg_files(directory)


# Print alle gevonden Markdown links
for figs in pdf_links:
    print(figs)
```

Figshrinker

> This page reuses BSD 3-Clause License content from TeachBooks (2024). ❝Find out more here.

```
# shrink file size of JPG
# file zoekt naar jpg bestanden en comprimeert deze tot een gewenste formaat.

from PIL import Image
import os

def compress_image(file_path, output_quality=50):
    """ Compress the image by changing its quality. """
    picture = Image.open(file_path)
    # You can also resize the image if needed using picture.resize()
    picture.save(file_path, "JPEG", optimize=True, quality=output_quality)

def find_and_compress_images(directory, max_size=1.5*1024*1024):
    """ Find all jpg files exceeding max_size and compress them. """
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.lower().endswith(".jpg"):
                full_path = os.path.join(root, file)
                if os.path.getsize(full_path) > max_size:
                    print(f"Compressing: {full_path}")
                    compress_image(full_path)

# Specify the root directory to start from
root_directory = 'I:/Book/2023'
find_and_compress_images(root_directory)
```

References

Hall, M. (2021). Which open licence should i choose? Online; posted 17-February-2021. CC BY. URL: https://agilescientific.com/blog/2021/2/17/which-open-licence-should-i-choose

Idema (Delft University of Technology), T. (2024). Latex to markdown conversion. Retrieved December, 2024. BSD-3. URL: https://gitlab.tudelft.nl/opentextbooks/latex-to-markdown-conversion

Moore, J. (2023). Learn multibody dynamics, sympy. URL: https://moorepants.github.io/learn-multibody-dynamics/sympy.html

Preston-Werner, T. (2025). Semantic versioning specification. Retrieved February 2025. CC BY 3.0. URL: https://semver.org/

TeachBooks (2024). Custom launch buttons. Retrieved December, 2024. BSD-3-Clause. URL: TeachBooks/Sphinx-launch-buttons

TeachBooks (2024). Git workflow: share content between book. Retrieved December, 2024. CC BY 4.0. URL: TeachBooks/Nested-Books

TeachBooks (2024). Github reusable action: publish your book online to github pages. Retrieved December, 2024. BSD-3-Clause. URL: TeachBooks/deploy-book-workflow

TeachBooks (2024). Local annotator extension. Retrieved December, 2024. BSD-3-Clause. URL: TeachBooks/annotator

TeachBooks (2024). Sphinx named colors. Retrieved January 2025. MIT. URL: TeachBooks/Sphinx-Named-Colors

TeachBooks (2024). Sphinx extension: grasple. Retrieved December, 2024. MIT. URL: TeachBooks/Sphinx-Grasple-public

TeachBooks (2024). Sphinx extension: image inverter. Retrieved December, 2024. BSD-3-Clause. URL: TeachBooks/Sphinx-Image-Inverter

TeachBooks (2024). Sphinx extension: jupyterbook-patches. Retrieved December, 2024. BSD-3-Clause. URL: TeachBooks/JupyterBook-Patches

TeachBooks (2024). Sphinx extension: tu delft theme. Retrieved December, 2024. MIT. URL: TeachBooks/Sphinx-TUDelft-theme

TeachBooks (2024). Sphinx extension: download link replacer. Retrieved December, 2024. BSD-3-Clause. URL: TeachBooks/Download-Link-Replacer

TeachBooks (2024). Teachbooks sphinx extension: rich hover over tips. Retrieved December, 2024. MIT. URL: TeachBooks/teachbooks-sphinx-tippy

TeachBooks (2024). Useful python code. Retrieved December, 2024. BSD-3-Clause. URL: TeachBooks/Useful_python_code

TeachBooks (2024). Your first teachbook using the github template. Retrieved December, 2024. CC BY 4.0. URL: TeachBooks/template

TeachBooks (2025). Sphinx prime applets. Retrieved April 2025. MIT. URL: TeachBooks/Sphinx-PRIME-applets

TeachBooks (2025). Sphinx iframes. Retrieved April 2025. MIT. URL: TeachBooks/sphinx-iframes

TU Delft Library (2024). Copyright information point. Retrieved December, 2024. CC BY 4.0. URL: https://www.tudelft.nl/library/support/copyright

Credits and License

Contents

You can refer to this book in its entirety as:

> TeachBooks Development Team (2024), *TeachBooks Manual*. TeachBooks/manual. Retreived [Month, Year]. CC BY 4.0.

This book is registered on Zenodo too: DOI 10.5281/zenodo.15100848

The introduction, structure and decisions on content is done under direction of the TeachBooks Development Team. The content is written by many contributors, of which there are many.

We anticipate that the content of this book will change continuously. Therefore, if you'd like to refer to a specific chapter, we recommend using the source code directly with the citation that refers to the GitHub repository and lists the date and name of the file. Although content will be added over time, chapter titles and URL's in this book are expected to remain relatively static. You can refer to individual chapters or pages within this book as:

> `<Title of Chapter or Page>`. In *TeachBooks Manual*. TeachBooks/manual (chapter `./book/intro/`, retrieved [`<date>`]).

Better yet, include a link to the specific commit! *If you know what this means, we assume you know how to do it.*

Acknowledgements

This book has many contributors, many of whom are also key members of the TeachBooks Team:

- Tom van Woudenberg
- Robert Lanzafame
- Freek Pols
- Dennis den Ouden-van der Horst
- Julie Kirsch
- Caspar Jungbacker

All from Delft University of Technology.

This doesn't include small contributions from various people from within and outside Delft University of Technology.

A better way to see the contributions is to check the Contributors Page of the GitHub repository.

A big "thank you" is also due to the various colleagues and teaching assistants that have worked as part of the TeachBooks Team for developing tools and providing support to improve this book, as well as the faculty of Civil Engineering and Geosciences and the Library at Delft University of Technology for significant financial support. The faculties of of Applied Sciences and Electrical Engineering, Mathematics and Computer Science have also provided support in the form of contributions from employees and student assistants..

License

External resources

Parts of this book are taken from other external resources and reused in various ways. If an author is not listed on a particular page, it is by the Authors, except as follows:

The following pages are included directly from an external resource and is not edited by the TeachBooks Team:

- page Local Annotator Extension. Original content licensed under BSD 3-Clause license.
- Page Releasing book online. Original content licensed under BSD 3-Clause license.
- Page Download link replacer. Original content licensed under BSD 3-Clause license.
- Page JupyterBook-Patches. Original content licensed under BSD 3-Clause license.

- Page Share content between books using submodules. Original content licensed under CC BY 4.0 license.
- Page Grasple. Original content licensed under MIT license.
- Page Image Inverter. Original content licensed under BSD 3-Clause license.
- Page Multilingual book. Original content licensed under BSD 3-Clause license.
- Page TU Delft theme. Original content licensed under MIT license.
- Page Rich hover over tips. Original content licensed under MIT license.
- Page TeachBooks template. Original content licensed under CC BY 4.0 license.
- Page Custom named colors. Original content licensed under MIT license.
- Pages Extensionchecker, filedownloader and figreturner, Figshrinker and Snippit. Original content licensed under BSD 3-Clause license.

The following pages contain content written by others, part of has been reused and/or modified by the TeachBooks Team:

- Pages Copyright and Licenses checklist and Copyright Considerations include text from TU Delft Library (2024) that is used in the copyright checklist items. Original content licensed under CC BY.
- Page Licenses: Figure 86 is included unmodified, along with paraphrased and quoted text (including hyperlinks) from Hall (2021). Work is licensed under CC BY.
- Page Licenses: Figure 87 is included unmodified. Work is provided by Kennisland under a CC0 license. Accessed at on December 17, 2024.
- page TeachBooks Versioning: includes text from Preston-Werner (2025) that has been modified. Original content licensed under CC BY 3.0.

Changelog

Contents

v1.1.0

Various updates with new and updates tooling, exercises and documentation. The full changelog is available here: TeachBooks/manual

v1.0.0

First release

Contact

Contents

Contact

If you encounter any issues report it by clicking on the top right corner of this page.

If you have questions, contact the editors at info@teachbooks.io.

Contribute

This book will continue to develop, so feel free to contribute TeachBooks/manual! You can do so directly by forking this repository and creating a pull request.

The released book can be found on on https://teachbooks.io/manual. This page shows the built book of the `release` branch. All branches will also be visible online, as pointed to in the actions summaries: TeachBooks/mirror_teachbook_manual

Some parts of this book are taken from other sources in the form of submodules (linked in the folder book/external). To contribute to those pages, contribute to the source repository directly with a fork and merge/pull request. If you intent to clone this book including its submodules, clone using: `git clone --recurse-submodules git@github.com:TeachBooks/manual.git`