The PreTeXt Guide

The PreTeXt Guide

Robert A. Beezer
University of Puget Sound

David Farmer
American Institute of Mathematics

Alex Jordan
Portland Community College

Mitchel T. Keller
Morningside College

March 11, 2024

Preface

Introduction. This part is the place to begin if you are new to PreTeXt. Chapter 1 is the introduction, overview, and philosophy. Then Chapter 2 intends to get you started quickly by showing how to set up a PreTeXt authoring environment and converting a document to HTML and LaTeX output formats. Notice that there are three parts which target different roles: the *Author's Guide* (Part II), the *Publisher's Guide* (Part ??) and the *Developer's Guide* (Part ??).

Author's Guide. This guide will help you *author* a PreTeXt document. So it serves as a description of the PreTeXt XML vocabulary, along with the mechanics of creating the source and common output formats. Chapter 3 is meant to be a short overview of the majority of PreTeXt's features, which can be skimmed to get a sense of PreTeXt's capabilities. Or it can be read quickly as you begin authoring and you can return as you need certain features. The roughly parallel Chapter 4 is much more comprehensive and is the first place to go for details not addressed in the overview. Note that the *Author's Guide* is not concerned with *publishing* your document, which is described in the *Publisher's Guide*.

Basic Reference. This part provides a quick overview of the minimal syntax for a variety of key PreTeXt features. Unlike the sample article, which is designed to demonstrate and stress test all aspects of PreTeXt, this guide will illustrate only the key elements of some of the most universally-used features of the language. In many cases, in addition to features not discussed, there may be alternative structures that are not given here.

Publisher's Guide. Even if you intend to distribute your document with an open license, and you are both author *and* publisher, it is still helpful and instructive to understand, and separate, the two different steps and roles. So visit this part of the *Guide* to learn how you can present, distribute, and maintain what you have authored.

Developer's Guide. This part provides advice, suggestions, and conventions for contributing to PreTeXt. For anything not answered here please use the `pretext-dev`[1] Google Group. Make a membership request and it will be processed quickly.

Appendices. In addition to the usual items you might expect in the back matter, such as an open license, glossary, references, and an index, there are numerous more specialized additions, mostly describing the installation of, or effective use of, various technical tools that are independent of PreTeXt (but useful or necessary).

[1] `groups.google.com/forum/#!forum/pretext-dev`

Contents

Part I

Introduction

Chapter 1

Why PreTeXt?

Welcome to "the Guide" for PreTeXt. You are likely eager to get started, but familiarizing yourself with this chapter should save you a lot of time in the long run. We will try to keep it short and at the end of early chapters we will guide you on where to go next. Not everything we say here will make sense on your first reading, so come back after your first few trial runs. When you are ready to seek further help, or ask questions, please read the Welcome to the PreTeXt Community in Appendix **??**.

1.1 Philosophy

PreTeXt is a **markup language**, which means that you explicitly specify the logical parts of your document and not how these parts should be displayed.

This is very liberating for an author, since it frees you to concentrate on capturing your ideas to share with others, leaving the construction of the visual presentation to the software. As an example, you might specify the content of the title of a chapter to be `Further Experiments`, but you will not be concerned if a 36 point sans-serif font in black will be used for this title in the print version of your book, or a CSS class specifying 18 pixel height in blue is used for a title in an online web version of your book. You can just trust that a reasonable choice has been made for displaying a title of a chapter in a way that a reader will recognize it as a name for a chapter. (And if all that talk of fonts was unfamiliar, all the more reason to trust the design to software.)

You are also freed from the technical details of presenting your ideas in the plethora of new formats available as a consequence of the advances in computers (including tablets and smartphones) and networks (global and wireless). Your output "just works" and the software keeps up with technical advances and the introduction of new formats, while you concentrate on the content of your book (or article, or report, or proposal, or ...).

If you have never used a markup language, it can be unfamiliar at first. Even if you have used a markup language before (such as HTML, Markdown, or basic LaTeX) you may need to make a few adjustments. Most word-processors are WYSIWYG ("what you see is what you get"). That approach is likely very helpful if you are designing the front page of a newspaper, but not if you are writing about the life-cycle of a salamander. In the old days, programs like `troff`[1] and its predecessor, `RUNOFF`[2] (1964), implemented simple markup languages to allow early computers to do limited text-formatting. Sometimes the old ways are the best ways.

PreTeXt is what is called an **XML application** or an **XML vocabulary** (I prefer the latter). That is, the source you write is "marked up" as XML, with specific **tags** that describe the semantic structure of your document. Authoring in XML might seem cumbersome at first, since some content will require more characters of markup than of content. Much of this markup can be quickly produced with a modern text editor, but it can still be overwhelming. We believe you will eventually appreciate the long-run economies, so keep an open mind. And if you are already familiar with XML, realize we have been very careful to design this vocabulary with human authors foremost in our mind.

[1]`en.wikipedia.org/wiki/Troff`
[2]`en.wikipedia.org/wiki/TYPSET_and_RUNOFF`

Principles. The creation, design, development, and maintenance of PreTeXt is guided by the following list of principles. These will become more understandable as you become more familiar with authoring texts with PreTeXt and should amplify some of the previous discussion.

List 1.1.1 PreTeXt Principles

1. PreTeXt is a markup language that captures the structure of textbooks and research papers.
2. PreTeXt is human-readable and human-writable.
3. PreTeXt documents serve as a single source which can be easily converted to multiple other formats, current and future.
4. PreTeXt respects the good design practices which have been developed over the past centuries.
5. PreTeXt makes it easy for authors to implement features which are both common and reasonable.
6. PreTeXt supports online documents which make use of the full capabilities of the Web.
7. PreTeXt output is styled by selecting from a list of available templates, relieving the author of the burden involved in micromanaging the output format.
8. PreTeXt is free: the software is available at no cost, with an open license. The use of PreTeXt does not impose any constraints on documents prepared with the system.
9. PreTeXt is not a closed system: documents can be converted to LaTeX and then developed using standard LaTeX tools.
10. PreTeXt recognizes that scholarly documents involve the interaction of authors, publishers, scholars, curators, instructors, students, and readers, with each group having its own needs and goals.
11. PreTeXt recognizes the inherent value in producing material that is accessible to everyone.

1.2 Understanding Your Source

Almost all of your time authoring in PreTeXt will be spent editing your **source** files. We now briefly describe what these files will look like and how to edit them.

File Format and Text Editors. Your source will be plain text **ASCII files**, which you create and edit with any number of text editors. Files can be saved with the `.ptx` extension, which might tell your text editor what sort of file you are editing and will provide syntax highlighting and code completion, among other features. If your editor does not recognize `.ptx`, then you can use the `.xml` extension which has wider editor support (but with fewer PreTeXt-specific features).

Popular text editors include Visual Studio Code, Sublime Text, vi, emacs, Notepad, Notepad++, Atom, TextWrangler, and BBEdit. But in particular, you should not use word processing programs like Word, LibreOffice, Google Docs, WordPerfect, AbiWord, Pages, or similar programs. Sometimes these editors are known as a **programmer's editor** (though we will be doing no programming). Support for writing HTML sometimes translates directly to good support for XML.

Visual Studio Code has support for PreTeXt documents via a free extension, and the editor is open source and cross-platform (Windows, OS X, and Linux). The developers of PreTeXt have also had a very good experience with Sublime Text, which is cross-platform, and can be used for free, though it has a very liberal paid license if you want to avoid nagging.

There are **XML editors**, which might be too complex for authoring in PreTeXt. They do have some advantages and XML Copy Editor is one that you might find useful.

Some text editors (like VS Code) have spell checking extensions. More generally, recommendations for a spell checker can be found in Section **??**.

Structure of your Source. If you start to think about the structure of a document (like an article or book) you will quickly realize that components are like blocks, stacked inside or next to other blocks. From the *outside* to *inside*, a book will have a number of chapters (next to each other, but all inside the book), and each might have sections (adjacent but inside the chapter). In the section, there will be a title, paragraphs, images, examples, theorems, and so on. Examples will themselves contain paragraphs. A theorem might contain a statement, which contains some paragraphs, which might contain some displayed math, and adjacent to the statement, there could be a proof, itself containing paragraphs, etc.

The hierarchical nature of XML is perfect to capture the hierarchical nature of a scholarly document. Consider the start of a PreTeXt document shown in Listing 1.2.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<pretext>
    <book>
        <title>Hello world!</title>
        <chapter>
            <title>Getting Started</title>
            <p>Welcome to PreTeXt!</p>
            <!-- TODO: find something more to say... -->
        </chapter>
    </book>
</pretext>
```

Listing 1.2.1 Source of a simple PreTeXt book project.

The first line is boilerplate that lets various programs know the rest of the file is XML, and the line start `<!--` is an example of a comment that won't appear in the output. Besides this, you can start to see how the structure of the book is layed out.

Whitespace and Indentation. The term **whitespace** refers to characters you type but typically do not see. For us they are **space**, **non-breaking space**, **tab** and **newline** (also known as a "carriage return" and/or "line feed"). Unlike some other markup languages, PreTeXt *does not ever use whitespace* to convey formatting information.

However, it can be useful to use whitespace to indent the different levels of the XML (and document) hierarchy. Use two (or four) spaces for indentation; a good editor will visually respect this indentation, and help you with maintaining the right indentation with each new line. Line up opening and closing tags at the same level of indentation, and your editor should let you "fold" the code to visually hide blocks.

Whatever you do, use a style and stick with it. You could put titles on a new line (indented) after creating a new chapter or section; some people like them on the same line, immediately adjacent. You could put a single blank line before each new paragraph, but not after the last. And so on. The choice is yours, but consistency will pay off when you inevitably come back to edit something. You have put a lot of work and effort into your source. You will be rewarded with fewer problems if you keep it neat and tidy.

In some parts of a PreTeXt document, every single whitespace character is important and will be transmitted to your output, such as in the `<input>` and `<output>` portions of a `<sage>` element. Since Sage code mostly follows Python syntax, indentation is important and leading spaces must be preserved. But you can indent all of your code to match your XML indentation and the entire `<input>` (or `<output>`) content will be uniformly shifted left to the margin in your final output.

Never use tabs, they can only cause problems. You should be able to set your editor to translate the tab key to a certain number of spaces, or to translate tabs to spaces when you save a file (and these behaviors are useful). Most editors have a setting that will show whitespace as a small faint dots or arrows, so you can be certain there is no stray whitespace *anywhere*.

Learn to Use Your Editor. Because XML requires a closing tag for every opening tag, it feels like a lot of typing. The VS Code PreTeXt extension comes with many **snippets** (code completions) that can fill out lots of the markup for you. More generally, any editor should know what tag to close next and there should be a simple command to do that (for example, in Sublime Text on Linux, `Alt-Period` gives a closing tag). Not only is this quick and easy, it can help spot errors when you forget to close an earlier tag.

If your editor can predict your opening tag, all the better. VS Code can recognize what tags are allowed at a given position. Sublime Text recognizes if you already have a `<section>` elsewhere, so when you start a second section, you very quickly (and automatically) get a short list of choices as you type, with the one you want at the top of the list, or close to it.

Invest a little time early on to learn, and configure, your editor and you can be even more efficient about capturing your ideas with a minimum of overhead and interference.

Revision Control. If you are writing a book, or if you are collaborating with co-authors, then you owe it to yourself and your co-authors to learn how to use revision control, which works well with PreTeXt since the source is just text files. The hands-down favorite is `git`. To fully understand it is beyond the scope of this guide but some information is provided in Appendix **??** which has hints on how to best use `git` together with a PreTeXt project.

If you use the workflow recommended in the Chapter 2 using GitHub's codespaces, you will get revision control via git automatically, and VS Code provides a graphical user interface for all the basic operations you need.

1.3 Converting Your Source to Output

Once you have content created in PreTeXt files (i.e., XML files), you will want to convert these files into a output format such as HTML, to be viewed in a web browser, or a PDF. Instructions for doing this will be discussed in Chapter 2, and in even more detail in Chapter **??**. Here we provide an overview of how the conversion works to help you understand what is possible.

With PreTeXt "installed" (on your computer or in the cloud), converting PreTeXt XML into a full HTML website can be as simple as typing `pretext build html` in a terminal, or hitting `Ctrl+Alt+B` in VS Code. Behind-the-scenes, these commands read through your XML and use **XSL** 1.0 (**eXtensiible Stylesheet Language**) to *transform* the XML source, using a number of XSL stylesheets that come with PreTeXt.

The recommended workflow for processing your source uses a python program we call the *PreTeXt-CLI* (CLI is **command line interface**). There are also a number of other free tools that can processes XML with XSL. For example, `xsltproc` is a command line program that is usually installed by default on Linux systems and MacOS. This was the recommended method in the early days of PreTeXt, and still works. Documentation for how to use `xsltproc` with PreTeXt can be found in Chapter **??**, but unless you are helping with the development of PreTeXt or are trying to do something fancy, you probably don't need it.

Some features of PreTeXt, such as the inclusion of images described in source, or including WeBWorK exercises, requires the use of an additional processing, done in python. Some of these also require additional software (such as LaTeX or Sage). The PreTeXt-CLI does this automatically when building (and regenerates these assets if they have changed since the last build). There is also a python script that can accesses these functions directly for use in development. See Chapter **??** if you are curious.

1.4 Where Next?

To start playing with PreTeXt right away, work through the Chapter 2. It will guide you through a cloud-based setup (no software install required) and you will create, edit, convert, and deploy your first document.

If you would like a general, high-level overview of features skip ahead to Chapter 3.

In-depth, comprehensive use of features is in Chapter 4.

If you have an existing project authored in LaTeX you may be interested in the conversion process described in Appendix **??**.

Chapter 2

Getting Started Tutorial

This chapter serves as a tutorial for quickly getting started with PreTeXt in your web browser using free services provided by GitHub[1]. (Advanced users who'd prefer to install our free and open-source software to their own machine may choose to skip ahead to Section 2.4.)

Objectives

At the end of this tutorial you will have...

- Created a free GitHub account.
- Created a GitHub Repository and Codespace for authoring PreTeXt in your web browser.
- Learned the first steps to editing a PreTeXt document.
- Converted your document to both LaTeX and accessible HTML.
- Deployed your HTML to the web via GitHub Pages.

The community does its best to keep this guide updated, but for even more up-to-date advice, join us at our regular Zoom drop-ins announced at our Google group[2] or watch a recording posted in Section 2.3.

2.1 Using GitHub

2.1.1 What is GitHub?

GitHub is a freely-available service for authoring, sharing, and deploying documents and source code, owned by Microsoft. It uses the free and open-source **Git** software for version management.

There are other services such as CoCalc[1] (see Section **??**) and GitLab[2] for managing PreTeXt documents online, as well as other ways to write PreTeXt that don't require anything besides installing the free and open-source PreTeXt software onto your own device (see Section 2.4 to learn more).

We will use GitHub for this tutorial as it the most popular way to share and disseminate PreTeXt documents, and provides the easiest pathway to getting started writing in the PreTeXt language.

To create your free GitHub account, follow the instructions on GitHub's signup page[3]. You can also log into an existing GitHub account if you already have one. Be sure to note your GitHub username and password in your password manager (or however you usually keep track of login credentials).

[1] `github.com`
[2] `groups.google.com/g/pretext-announce/`
[1] `cocalc.com`
[2] `about.gitlab.com/`
[3] `github.com/signup`

Tip! Educators and non-profit researchers can get many of GitHub's paid features for free. While this is not strictly required for the rest of the tutorial, it's a useful way to increase GitHub's free Codespaces usage quotas.

Apply at Education.GitHub.com[4] to unlock these features. In our experience, applications are usually processed quickly for `.edu` email addresses, but you do not need to wait for approval to continue on with this tutorial.

2.1.2 Three GitHub concepts

This tutorial uses three GitHub services:

Codespaces (`github.dev`) The Codespace for your project is an application run in your web browser that gives you access to a virtual computer with all the software recommended to author PreTeXt installed for you automatically. This Codespace is private to you, and lives at an address like `https://username-random-words-abc123.github.dev`.

Repository hosting (`github.com`) The **repository** for your project represents the history of its edits that have been "committed and synced" from your Codespace to it. This repository can be public or private (though we encourage public repositories as they help the community provide support for each other), and lives at an address like `https://github.com/username/reponame/`.

GitHub Pages (`github.io`) The **GitHub Pages** service provides free hosting for websites such as the HTML generated from a PreTeXt project. This website is public, and lives at an address like `https://username.github.io/reponame/`.

Broadly speaking, you "author" within your Codespace, which you periodically "commit and sync" to your repository, and then occassionally "deploy" to your public GitHub Pages website.

2.1.3 Creating your repository and Codespace

Follow the instructions at `https://github.com/PreTeXtBook/pretext-codespace` to get started creating your repository and Codespace. You'll have the option to make your repository public (recommended if you want support from the rest of the PreTeXt community) or private. Either way, those instructions will also walk you through creating your private Codespace for authoring.

This takes a few moments, but is a one-time process. Take note of the `github.com` URL your new repository lives at so you can find it the next time you want to work on your project. (You can always access your `github.dev` Codespace link from there via the Code menu.) Then you'll be ready for Section 2.2.

2.2 Your First PreTeXt Document

At this point, you should have a PreTeXt project set up as a `github.com` repository with a `github.dev` Codespace. You can use the Code menu on the repository webpage to pull up the Codespace environment in your web browser if you haven't already.

The left-hand menu should display a file tree, containing a folder called `source` with a file called `main.ptx`. These files were created when you set up your Codespace, and form a complete, albeit very short, PreTeXt document.

Now you are ready to build it!

2.2.1 Building for web

You can build your entire project in a few different ways.

- Click the "PreTeXt" button in the center left of the bottom toolbar of the VS Code window (see Figure 2.2.1). A dialog will pop up asking which PreTeXt command you want to run. Select `Build` to get a menu of options to select a **target** to build: choose `web`.
- You can use the keyboard shortcut CTRL+ALT+p (replacing CTRL with CMD if you have a Mac) to get the same dialogs. Or to build in one step, use CTRL+ALT+b.

[4]`education.github.com/discount_requests/pack_application`

- Select a PreTeXt command from the VS Code command pallette, which you can access by clicking the icon in the bottom left of the VS Code window. You can also access this by typing CTRL+SHIFT+p (again, replacing CTRL with CMD if you have a Mac). Start typing "pretext" to get a list of commands available.

- If you are comfortable entering commands in a terminal/command prompt, you can access one in your Codespace using CTRL+`. Then you can run `pretext build web` to build your project.

The resulting HTML files will be available in the `output/web` directory of your project. However, to view it, you should NOT navigate there and open the files. Instead, read on.

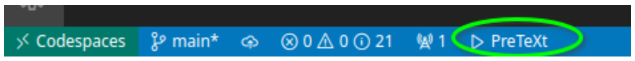


Figure 2.2.1 PreTeXt commands in Codespaces

2.2.2 Viewing

You can preview these HTML files you just built using the **View** command. Again, you can access this in multiple ways: PreTeXt buttom in the toolbar, CTRL+ALT+p, etc. Select `View` from the dialog. You may be given options on how to view the document, depending on what VS Code plugins you have available to you. Try one or another until you're able to view your web build in either a new tab of your browser or a tab within VS Code.

The VS Code Live Preview is a good option, but it is buggy when used inside Codespaces. It seems to help to use the VS Code command pallette to run `Live Preview: Show Preview (External Browser)`, then close the tab that opens, and start the process over. You may need to do this a few times before it works.

Now is a great time to try to make edits to your source files (maybe change the title). Note that these changes aren't updated live in your preview: you will need to build again, and then refresh the preview window to see them. Note, you do not need to run the `View` command again unless you stop the preview server.

2.2.3 Building for print

The instructions above can be repeated to produce LaTeX code: just choose `print-latex` instead of `web` as your target. The resulting files are available in `output/print-latex`.

Of course, it'd be even more convenient to produce a PDF directly. This requires software that can process LaTeX, which should be installed in the PreTeXt Codespace by default. Repeat the above instructions with the `print` target to produce a PDF. It can be downloaded by right-clicking `output/print/main.pdf` in the VS Code file explorer, or previewed using a View command.

2.2.4 Saving your work

Using Codespaces will keep all your files "in the cloud", saved automatically as you edit. As long as your Codespace is active, your files will be saved there for your private use. However, inactive Codespaces are periodically cleaned up by GitHub (as of writing, this happens after one month of inactivity), so you'll need to periodically **commit & sync** your work to your repository where it will never be deleted.

Recall that your Codespace lives at `github.dev`, while your repository has a `github.com` address like `https://github.com/userna`
This repository serves as a backup of your work in the Codespace, and has the added benefit of allowing collaborators to access your files as well. As a bonus, if you made your repository public, members of the PreTeXt community who watch the PreTeXt-support[1] Google group can create their own Codespace based on your public repository and easily answer any questions you have.

While Git and GitHub have a lot of features, there's a very simple way to use them via Codespaces. As you edit files, you'll notice that their filenames will turn orange, and new files will appear green. Likewise, a blue number will appear in the left sidebar.

[1] `groups.google.com/g/pretext-support`

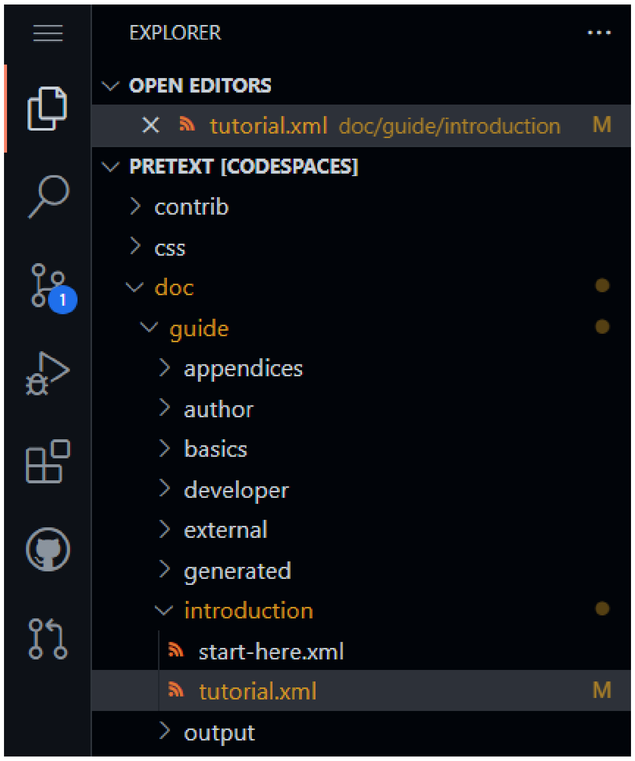
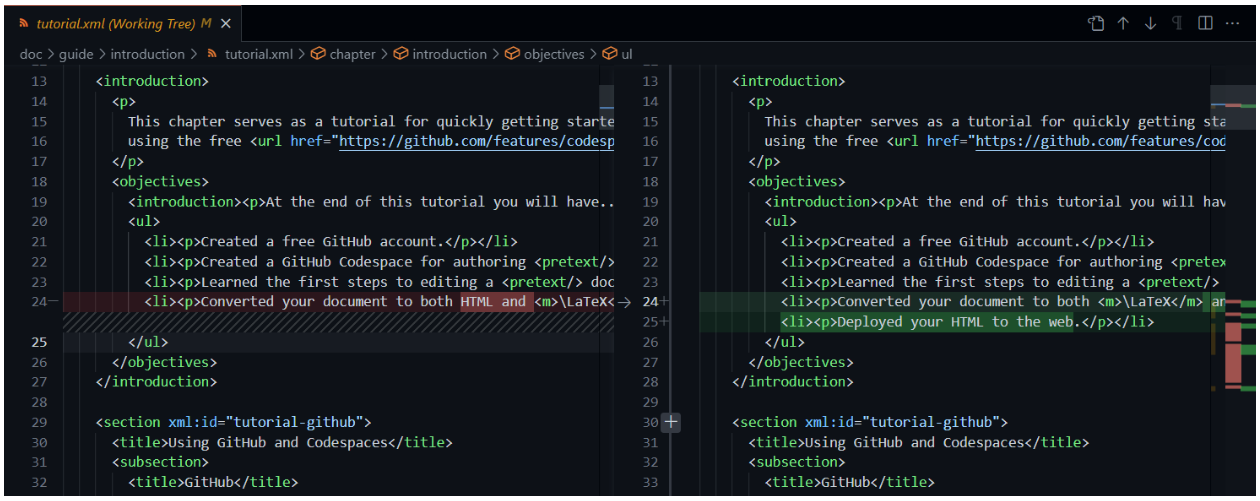


Figure 2.2.2 Filenames changing color as they are edited in Codespaces

This blue badge is next to the Source Control view. You will notice a list of files that were changed; you can click on any of these to see what the changes are.


tutorial.xml (Working Tree) M
doc > guide > introduction > tutorial.xml > chapter > introduction > objectives > ul

```
<introduction>
  <p>
    This chapter serves as a tutorial for quickly getting starte
    using the free <url href="https://github.com/features/codesp
  </p>
  <objectives>
    <introduction><p>At the end of this tutorial you will have..
    <ul>
      <li><p>Created a free GitHub account.</p></li>
      <li><p>Created a GitHub Codespace for authoring <pretext/>
      <li><p>Learned the first steps to editing a <pretext/> doc
      <li><p>Converted your document to both HTML and <m>\LaTeX<

    </ul>
  </objectives>
</introduction>

<section xml:id="tutorial-github">
  <title>Using GitHub and Codespaces</title>
  <subsection>
    <title>GitHub</title>
```

```
<introduction>
  <p>
    This chapter serves as a tutorial for quickly getting sta
    using the free <url href="https://github.com/features/cod
  </p>
  <objectives>
    <introduction><p>At the end of this tutorial you will hav
    <ul>
      <li><p>Created a free GitHub account.</p></li>
      <li><p>Created a GitHub Codespace for authoring <pretex
      <li><p>Learned the first steps to editing a <pretext/>
      <li><p>Converted your document to both <m>\LaTeX</m> ar
      <li><p>Deployed your HTML to the web.</p></li>
    </ul>
  </objectives>
</introduction>

<section xml:id="tutorial-github">
  <title>Using GitHub and Codespaces</title>
  <subsection>
    <title>GitHub</title>
```

Figure 2.2.3 A ``Git diff'' showing changes in a file

Type a message describing the changes you've made then click the green "Commit and Sync" button. If it just says "Commit", use the drop-down menu to choose "Commit and Sync". (If you forget to type a message describing the changes you've made, then a new tab will open: "COMMIT_EDITMSG" where you can type the message. When you are done, close the tab.)

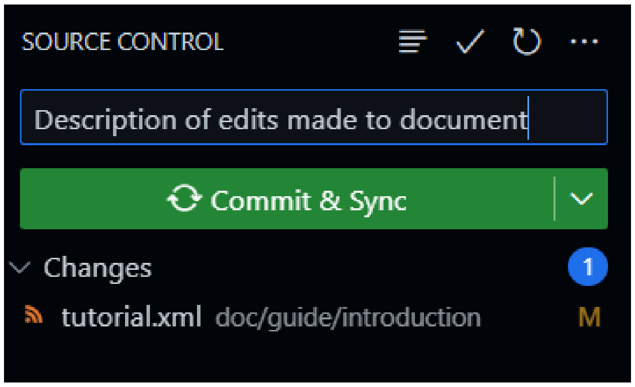

SOURCE CONTROL
Description of edits made to document
Commit & Sync
Changes
1
tutorial.xml doc/guide/introduction
M

Figure 2.2.4 Commiting and syncing changes

To see that this is successful, return to your `github.com` repository webpage. You should see your files with all your committed/synced changes. (That is, most of them: many files, such as log files and temporary build files that appear in gray within your Codespace, will not be synced. This is no problem: they are created during a build automatically and don't need to be, and really shouldn't be, saved or shared with others.)

2.2.5 Generating assets

If your document contains certain elements, you might need to **generate** their assets for use in certain output formats. Depending on your build target, these include:

- `<latex-image>`
- `<sagemath>`
- `<asymptote>`
- `<youtube>`
- `<webwork>`
- `<codelens>`

Starting in CLI version 1.7, these assets will be automatically generated whenever you build your output. If you change the source of these assets, they will be regenerated when you build.

Regardless of which version of the CLI you are using, you can generate assets as a separate step in much the same way you run a build. You will see a `Generate` option in the PreTeXt command dialog, just below `Build`. Select your target and wait for the process to complete, then `Build` once more to incorporate your generated assets.

2.2.6 Deploy

So you have worked tirelessly to prepare course notes or a book, built and previewed, synced changes to your git repository, and now you are ready to share the results of your efforts with the world. It's time to **deploy** your project!

From the "PreTeXt Commands" dialog, select "Deploy". This will automatically take the most recent build of your web target and host it through GitHub Pages[2]. Watch the output pane for a link to your published site; unlike the preview link you've been using on `github.dev` which is private to only you, this `github.io` link is ready to share with the world. (It can take a few minutes for the site to get set up or updated; there should be another link to view the progress of the GitHub "action" that reports the progress.)

By default, doing a deploy will just publish your web target. It is also possible to deploy multiple targets along with a "landing" page directing a visitor of your site to the different versions of your project. See Section **??** for more information.

[2]`pages.github.com/`

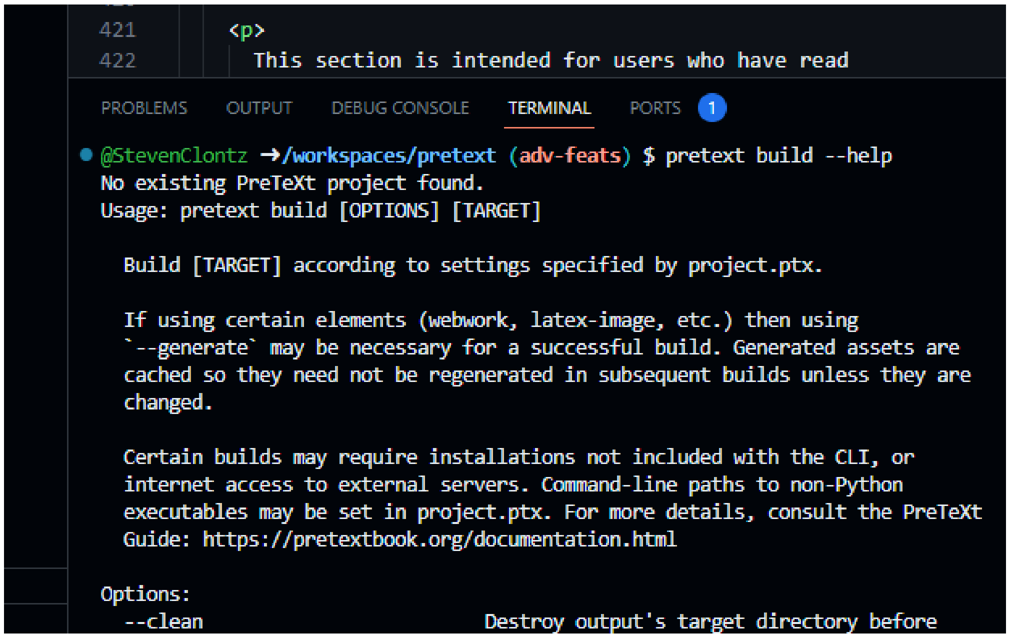
2.2.7 Using this guide and advanced features

The rest of this guide will help you on your way. However, keep in mind that this guide is the work of many volunteers over many years, and certain sections may assume the reader is using mechanisms for writing PreTeXt that have been around for much longer than the Codespaces environment recommended for this tutorial.

In particular, there are two advanced mechanisms used by many PreTeXt authors: the **PreTeXt developer script** Chapter **??** (i.e. the `pretext/pretext` script) and the **PreTeXt CLI** Section **??**.

Under the hood, the PreTeXt CLI is what you're using in Codespaces, and it also has the ability to call the PreTeXt developer script as well. If you ever want to use a PreTeXt CLI command, you can open a **Terminal** in your Codespace using the menus, or by pressing Ctrl+` (the backtick key, found in upper left of many keyboards).

From the terminal, you can type in any PreTeXt CLI commands directly. For example, typing in the CLI command `pretext build web` and running it by pressing Enter builds the web target.



```
    <p>
      This section is intended for users who have read
```

```
@StevenClontz ➜/workspaces/pretext (adv-feats) $ pretext build --help
No existing PreTeXt project found.
Usage: pretext build [OPTIONS] [TARGET]

  Build [TARGET] according to settings specified by project.ptx.

  If using certain elements (webwork, latex-image, etc.) then using
  `--generate` may be necessary for a successful build. Generated assets are
  cached so they need not be regenerated in subsequent builds unless they are
  changed.

  Certain builds may require installations not included with the CLI, or
  internet access to external servers. Command-line paths to non-Python
  executables may be set in project.ptx. For more details, consult the PreTeXt
  Guide: https://pretextbook.org/documentation.html

Options:
  --clean                         Destroy output's target directory before
```

Figure 2.2.5 Using the PreTeXt CLI with a Codespaces terminal

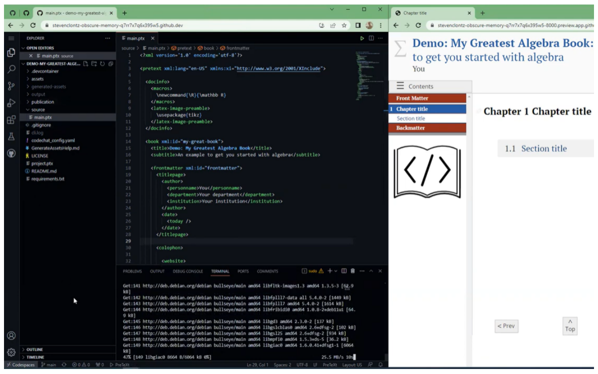
The CLI should be sufficient to do nearly everything you want to do for your project, and using the developer script should be exercised with caution. Nonetheless, to access a `pretext/pretext` developer script feature, you can use `pretext devscript`. For example, if the documentation suggests a command like `pretext/pretext -foo bar`, you could try running `pretext devscript -foo bar`.

2.3 Videos

Regularly produced videos showcasing how to get started with PreTeXt as the ecosystem evolves over time is curated by Steven Clontz[1]. Roughly once a month, a "Getting Started with PreTeXt" presentation is offered to new or prospective community members, showcasing the latest recommendations for writing modern PreTeXt.

The most recent recording of this series is provided here for your reference.

[1] `clontz.org`



```
<?xml version='1.0' encoding='utf-8'?>

<pretext xml:lang="en-US" xmlns:xi="http://www.w3.org/2001/XInclude">

  <docinfo>
    <macros>
      \newcommand{\R}{\mathbb R}
    </macros>
    <latex-image-preamble>
      \usepackage{tikz}
    </latex-image-preamble>
  </docinfo>

  <book xml:id="my-great-book">
    <title>Demo: My Greatest Algebra Book</title>
    <subtitle>An example to get you started with algebra</subtitle>

    <frontmatter xml:id="frontmatter">
      <titlepage>
        <author>
          <personname>You</personname>
          <department>Your department</department>
          <institution>Your institution</institution>
        </author>
        <date>
          <today />
        </date>
      </titlepage>

      <colophon>

        <website>
```

```
Get:141 http://deb.debian.org/debian bullseye/main amd64 libfltk-images1.3 amd64 1.3.5-3 [62.9
kB]
Get:142 http://deb.debian.org/debian bullseye/main amd64 libfplll7-data all 5.4.0-2 [1449 kB]
Get:143 http://deb.debian.org/debian bullseye/main amd64 libfplll7 amd64 5.4.0-2 [1614 kB]
Get:144 http://deb.debian.org/debian bullseye/main amd64 libfribidi0 amd64 1.0.8-2+deb11u1 [64.
9 kB]
Get:145 http://deb.debian.org/debian bullseye/main amd64 libgd3 amd64 2.3.0-2 [137 kB]
Get:146 http://deb.debian.org/debian bullseye/main amd64 libgslcblas0 amd64 2.6+dfsg-2 [102 kB]
Get:147 http://deb.debian.org/debian bullseye/main amd64 libgsl25 amd64 2.6+dfsg-2 [934 kB]
Get:148 http://deb.debian.org/debian bullseye/main amd64 libmpfi0 amd64 1.5.3+ds-5 [36.2 kB]
Get:149 http://deb.debian.org/debian bullseye/main amd64 libgiac0 amd64 1.6.0.41+dfsg1-1 [6064
kB]
47% [149 libgiac0 8664 B/6064 kB 0%]                                   25.5 MB/s 10s
```

Standalone

Figure 2.3.1 Getting Started in 2023 March

2.4 For Advanced Users New to PreTeXt

This section is intended for users who have read Chapter 1 and are already experienced with other open-source software projects or command-line tools. If you're more interested in a browser-based workflow to getting started with PreTeXt, check out Section 2.1.

PreTeXt is an open-source XML[1] language primarily powered by XSLT[2] and Python[3] tools.

For new authors comfortable working on the command line who want to use their favorite text editor or IDE, we recommend `pip installing` the PreTeXt-CLI from the Python Package Index. Section **??** has these details, and `pretext -h` and `pretext CMD -h` are at your disposal as well after installation.

Some context for experienced Python developers: PreTeXt development is primarily split over two GitHub repositories: PreTeXtBook/pretext[4] for the "core" functionality of PreTeXt, and the more recent PreTeXtBook/pretext-cli[5] that packages up these resources into a Python package with several UX enhancements such as a simplified command line interface and project management that does not require the use of custom makefiles.

If you're interested in potentially contributing back to PreTeXt someday, please feel free to request to join our developer Google Group[6] and say hello!

[1] en.wikipedia.org/wiki/XML
[2] en.wikipedia.org/wiki/XSLT
[3] en.wikipedia.org/wiki/Python_(programming_language)
[4] github.com/PreTeXtBook/pretext
[5] github.com/PreTeXtBook/pretext-cli
[6] groups.google.com/g/pretext-dev

Part II

Author's Guide

Chapter 3

Overview of Features

This chapter is a high-level view of the important concepts, features and design decisions that go into the creation of PreTeXt. For careful exact descriptions of details, we will direct you to one of the many sections in the Topics chapter. So this chapter should make you aware of what is possible and expand on the philosophy described earlier in Section 1.1, while also giving you examples of many basic constructions you can use to get started quickly.

3.1 Structure

A PreTeXt document is a nested sequence of **structural divisions**. For a book, these would go `<part>`, `<chapter>`, `<section>`, `<subsection>`, and `<subsubsection>`. Using `<part>` is optional, but a book must always use `<chapter>` (or else it is not a book!). No skipping over divisions. For example, you cannot divide a `<section>` directly into several `<subsubsection>`s without an intervening `<subsection>`.

An `<article>` starts divisions from `<section>`, though it may choose to have no divisions at all. `<paragraphs>` are exceptional. They lack a full set of features, but can be used to divide anything, in books or in articles, though they are always terminal since you cannot divide them further. You will have noticed that we prefer the generic term **division** (rather than "section") since a `<section>` is a very particular division.

A division may be unstructured, in which case you fill it with paragraphs and lists and figures and theorems and so on. But if you choose to structure a division it must look like the following:

- An optional `<introduction>`
- One or more divisions of the next finer granularity
- An optional `<conclusion>`.

Either version may have a single `<exercises>` division at the end, or other "specialized" divisions. The structured version may have more than one `<exercises>`, and more than one of each of the types of specialized divisions. For example a `<references>` is a second example of a specialized division. (See Section 4.7.)

The `<introduction>` and `<conclusion>` divisions are meant to be short, and may not contain any other numbered tag. No exercises, theorems, listings, etc. If you want to have an introductory division with any of the numbered elements you are free to omit the `<introduction>` and use the next finer subdivision with a `<title>` of "Introduction".

Every division tag can carry an `@xml:id` attribute, and it is a good practice to (a) provide one, (b) use a very short list of words describing the content, and (c) adopt a consistent pattern of your choosing. Do not use numbers, you may later regret it. These are optional, and with practice you will learn how best to use them. See Section 3.4 just below for more on this.

The `<exercises>` and `<references>` tags are special divisions, see Section 4.3 and Subsection 4.7.1.

This explanation is expanded and reiterated at Section 4.6 and is worth reading earlier rather than later.

3.2 Paragraphs

Once you have divisions, what do you put into them? Most likely, paragraphs. We use long, exact names for tags that are used infrequently, like `<subsubsection>`. But for frequently used elements, we use abbreviated tags, often identical to names used in HTML. So a paragraph is delimited by simply the `<p>` tag.

Lots of things can happen in paragraphs, some things can *only* happen in a paragraph, and some things are *banned* in paragraphs. Inside a paragraph, you can emphasize some text (`<em>`), you can quote some text (`<q>`), you can mark a phrase as being from another language (`<foreign>`), and much more. You can use almost any character your keyboard can produce, but need to be careful with the three XML exceptional characters: ampersand (&), less than (<), and rarely, greater than (>). (See Section 3.14.) You must put a list inside a paragraph, and all mathematics (Section 3.6) will occur inside a paragraph. You cannot put a `<table>` or a `<figure>` in a paragraph, and many other structured components are prohibited in paragraphs.

Paragraphs are also used as part of the structure of other parts of your document. For example, a `<remark>` could be composed of several `<p>`. As you get started with PreTeXt, remember that much of your actual writing will occur inside of a `<p>` and you will have a collection of tags you can use there to express your meaning to your readers.

So early in your writing project, familiarize yourself with the components of a paragraph detailed in Section 4.1.

3.3 Blocks

Besides paragraphs (Section 3.2) the most common object to include in a division is what we informally refer to as a **block**. These are self-contained units of text, almost always set-off visually, and likely with a number and a title. If you know LaTeX, you may be in the habit of calling these **environments**. Mathematical results are one example, and you can start at Section 3.20 to learn more. There are others that are more general-purpose, such as `<remark>` and `<example>`. While fundamentally different from these blocks that are textual with reflowable lines, objects like `<figure>` and `<table>` (Section 3.13) or `<program>` and `<console>` (Section 3.10) are blocks, even if their contents are more rigid or spatial. For a more precise description, see Section 4.2.

3.4 Cross-References

Cross-references in a PreTeXt document are easy, powerful and flexible. So it is worth familiarizing yourselves with them early, here and then ahead in Section 4.5.

Any element that you place a `@xml:id` on can become the target of a cross-reference. This could be a division, a remark, a bibliographic entry, or a figure. So for example, suppose your source had `<subsection xml:id="subsection-flowers">` and someplace else you wrote `<xref ref="subsection-flowers" />`. Then at the latter location you would get a reference to the Subsection that discusses flowers. In print this might just be the number for the subsection, but in various electronic output formats, these cross-references can be very powerful interactive ways to explore the content. And the mechanism is always the same, pair up an `@xml:id` on a target with a `@ref` on an `<xref>` cross-reference.

Since the value of an `@xml:id` is also used in a variety of ways, such as to construct some file names, some care should be taken in how you author them. We limit the possible characters to letters and numbers (`a-z`, `A-Z`, `0-9`), with hyphens and underscores (`-_`) available as word-separators. Our advice is to stick to lowercase letters, though we are not yet aware of any problems with case-insensitivity. So in short, use **kebab-case** or **snake-case** for your `@xml:id` values.

For more, see Section 4.5 because cross-references have many features. But first, here are two features you do not want to miss. In the early stages of writing, you can author `<xref provisional="subsection-flowers" />` to point to a subsection you are contemplating (but have not written yet) and you will get various polite reminders to get that straightened out eventually (see Section **??** for details). Also the default behavior is to automatically provide the generic name of the target, so you will get something like "Subsection 4.3.2" without ever typing the "Subsection" part. If you move the target, the generic name will adjust if necessary, and if you switch to one of the supported languages, the generic name will switch language (see `[provisional cross-reference: topic-on-languages]`.

3.5 Titles

Divisions always require titles, you accomplish this with a `<title>` tag first thing. Almost everything that you can use in a paragraph can be used in a title, but a few constructions are banned, such as a displayed mathematical equation (for good reason). Try to avoid using footnotes in titles, even if we have tried to make them possible.

Since titles migrate to other places, such as a Table of Contents, there are options for variants of a title, such as a short version, or a markup-free version. Some (major) titles may also be structured as a sequence of `<line>` elements to control line-breaks for long titles.

Many, many other structures admit titles. Experiment, or look at specific descriptions of the structure you are interested in. Titles are integral to PreTeXt, much like cross-references. Titles migrate to the Table of Contents, get used in page headers for print output, can be used in lists (such as a List of Figures), and can be used as the text of a cross-reference, instead of a number. You might be inclined to not give a `<remark>` a title, but it would definitely be good practice to do so (study Best Practice 4.8.1). For more details consult Section 4.8.

3.6 Mathematics

With experience, you may realize that PreTeXt utilizes three principal languages. One is the narrative of everyday sentences and paragraphs. Most of what you write in a paragraph, or a table cell, or a title, or a caption, or an index heading, is in this language. Then there is the structural language, which is the majority of the elements in PreTeXt, such as `<chapter>`, `<theorem>`, or `<figure>`. Then finally, there is the language of mathematical symbols and notation.

A key design decision is that mathematical symbols, expressions and equations are authored using LaTeX syntax. More precisely, we support the symbols and constructions provided by MathJax[1], which quite closely follows the `amsmath` package maintained by the American Mathematical Society. Neither you nor I want to write MathML[2] by hand!

The symbols and macros supported by MathJax can be found at their Supported LaTeX commands[3] documentation. Look here to see which parts of LaTeX may be used in your mathematical expressions.

For **inline** mathematics, use the short `<m>` tag within a `<p>` (or within a `<title>` or `<caption>`). For example, `<m>\alpha^2 + \beta^4</m>` will do what you expect, in print and in electronic outputs. To get a single equation, centered, with some vertical separation before and after, use the `<me>` tag ("math equation") in the same way within a `<p>`, but do not try using it within a `<title>`. For example, `<me>\rho = \alpha^2 + \beta^4</me>`. If you want your equation numbered, switch to the `<men>` tag (`n` = "numbered").

There is a way to incorporate your own (simple) custom LaTeX macros within mathematics (only). They will be effective in your print and electronic outputs, and can be employed in graphics languages like `tikz` and Asymptote. You can also author multi-line **display mathematics** using the `<md>` tag surrounding a sequence of `<mrow>` elements (or the `<mdn>` variant for numbered equations). We defer the details to Section 4.9.

3.7 Images

You can include an image via the `<image>` tag, using the `@source` attribute to provide a filename, likely prefixed by a relative path from the top-level of the appropriate directory. Read Section **??** for details on how to set these directories correctly. If you are starting a new project, using the PreTeXt-CLI (with the command `pretext new book`, for example), then most of the setup portion is done for you and the top-level directory for images that are created external to the project is called `assests`, and it is a sibling of the `source` directory. It is your responsibility to locate that file properly relative to this directory, and that the file format is compatible. So, for example, suppose your source contained `<image source="images/butterflies.jpg"/>`. Then you would want to have a directory named `images` below wherever you set the `@external` top-level directory, and you would place the `butterflies.jpg` file inside of the `images` directory.

[1] `www.mathjax.org/`
[2] `en.wikipedia.org/wiki/MathML`
[3] `docs.mathjax.org/en/latest/tex.html#supported-latex-commands`

The @width attribute can be used to control the size of the image. Widths are expressed as a percentage of the available width, such as width="60%". Instead of a width, you can also specify margins and the width will be deduced.

The optional @rotate attribute controls the angular rotation of the image about its center, for example image/@rotate="30" will rotate the image 30° counterclockwise.

You may want to wrap your image in a <figure> to have it centered, and to have some vertical separation above and below. A <figure> must also have a <caption>, and the figure will be numbered.

You can also place an "anonymous" image (no caption, no number) almost anywhere you might place a paragraph (but not *within* a paragraph). Note also that the <sidebyside> tag provides some very flexible options for placing several images (Section 3.19) together, or combining figures with subcaptions.

If you wish to construct technical diagrams, with editable source, and perhaps including the use of LaTeX macros, PreTeXt provides support for authoring with graphics languages such as Asymptote, TikZ, PGF, PSTricks, and xy-pic in addition to using Sage code to describe a plot or image. In most cases output can be obtained as smoothly-scalable SVG images, in addition to other formats like PDF or PNG. Making all this happen is one of the more technical aspects of PreTeXt, so read the details in Section 4.14 along with frequent references there to the pretext script described in Chapter **??**.

For accessibility, every <image> should either have a description or it should explicitly declare itself to be a decorative image setting @decorative to the value yes. Descriptions are of two types. A <shortdescription> should minimally describe the important information in the image that is not already present in the surrounding text. It should have no element children except possibly <var> children. This content will be used as the @alt attribute for the HTML <img>. Some screen readers may cut off reading this content after the first 100–140 characters, therefore you should keep this element short. A <description> element should be structured with <p> and <tabular> children and has no technical limit on length. It may describe the image more completely. The content of the <description> should describe the image, but should not be necessary for a sighted reader to take in the important features of the image. (If there are features of the image that need to be explained to all readers, this should happen in the main body of the text.) An <image> may have one or both types of description, but it is incorrect to have neither unless @decorative is set to value yes.

3.8 Lists

Ordered lists (numbered), unordered lists (bullets) and description lists (defined terms) are all supported, and syntax generally follows HTML. Lists usually live within a paragraph (<p>), though there are limited exceptions. Their structure is given by the <ol>, <ul>, <dl> tags (respectively). These can specify a variety of options for the labels via attributes, as described in Section 4.11.

List items, for any of the three types, are delimited with the <li> tag. What is different from HTML is that the contents of a list item may be structured, with paragraphs (<p>) being the most likely and frequent element. So to nest lists you begin a paragraph in a list item of the outer list, then begin the inner list within that paragraph. However, a simple list item may be authored just like you were authoring within a paragraph, much like writing sentences elsewhere. A structured list item may begin with <title> for ordered and unordered lists, and is mandatory for a description list. In this latter case, the text of the <title> will become the text that is being described (the label of the list item). For the optional uses, the title will be rendered as its own paragraph, with a different font (perhaps italics or oblique). A description list cannot be contained within another list. In other words, it is a "top-level" list.

Lists are more complicated than they appear, so be sure to read the details at Section 4.11 before you start designing really involved lists.

3.9 Exercises

Textbooks in many disciplines have exercises for the reader. In PreTeXt there are five places where you can set a question for the reader to pursue.

Divisional There is a special <exercises> division and an <exercise> placed there is then known as a **divisional exercise**. This division supports extra features designed for exercises, such as an <exercisegroup> for short exercises with common instructions. The <exercises> division can be used at any level. In other words, it can be a peer of any other division.

Inline Immediately within any division, you can interrupt the narrative with an **inline exercise**. It will be rendered similar to a <theorem> or other block, with a number, and a optional <title>.

Reading Question Another specialized division, <reading-questions>, can be used to house <exercise> designed to test or guide a reader's comprehension of the material in that division.

Worksheet The main component of a <worksheet> is an <exercise> (Section 3.11). Notably the exercises in a worksheet may be arranged with a <sidebyside> element (Section 3.19).

Project A <project> is similar to an inline exercise, other than the type name and the fact that it can run on a separate counter from theorems, figures, etc. (When running on a separate counter, the same counter is used for <activity>, <exploration>, and <investigation>.)

If an <exercise> is simply a statement of the question, then it may be authored with paragraphs (<p>) and similar elements. If an <exercise> has hints, answers, or solutions, then it must be structured with a <statement>, followed by (possibly several) optional <hint>, <answer> and/or <solution>. Conceptually, an <answer> is a short final result, while a <solution> provides details about the route to the answer. Each of these four components is structured further, with paragraph-like elements, and the exercise itself may have a <title>. A title is strongly encouraged for inline exercises, and nearly-mandatory if you plan to have inline exercises rendered in knowls in a conversion to HTML.

There is a wide variety of interactive exercise types you may specify, such as multiple choice, Parson problems, matching, and more. See Section 4.12 for descriptions of each type and the details of markup for each.

You need (and want) to have the hints, answers and solutions grouped with the statement as you author, but there is a lot of flexibility on making these available at the location of the exercise, or in the back matter. See Section 4.13 for more.

An inline exercise typically gets a fully qualified unique number and is rendered similar to an <example> or a <remark>. A divisional exercise (including reading questions and worksheet exercises) only gets a sequential number, though this can be overridden with the @number attribute if you want to maintain stable numbering in response to edits. (Be careful, once you override the sequential numbering, you probably need to manually specify every subsequent number, so save overrides for when your project matures.)

Within a run of divisional exercises a subgroup can be delimited as an <exercisegroup>, which requires an <introduction> and allows a <conclusion> to explain some commonality. A <title> is optional, and a default will be provided otherwise. An <exercisegroup> should be rendered in some way that makes it clear to the reader that they are a group.

3.10 Programs and Consoles

If you are writing about computer science, or more general scientific or engineering topics, you may wish to include sample computer programs, or command-line sessions. A <program> will contain a complete computer program, or a portion of the code from a program. Some <program> can execute in an online output format, while others are both editable and executable. This behavior depends on the language used and the host employed. A <console> holds a command-line session. These elements feature monospace fonts, preservation of whitespace, and syntax highlighting. These may also be placed in a <listing> to be more prominent, or as the target of a cross-reference. See Section 4.15 for details.

3.11 Worksheets

Another division is a <worksheet>. It is similar to a <chapter>, <section>, and so on, but with some variations to support a worksheet or in-class activity. Here we recognize the primacy of printed output (perhaps to bring into a classroom), and the online version is a less-capable representation.

There is no limit to what you can place in a <worksheet> division: objectives, an introduction, theorems, figures, images, and so on. But the principal element is an <exercise>, which mostly behaves like an <exercise> in an <exercises> division, but with additional capabilities.

An <exercise> in a <worksheet> can have a specified width when included in a <sidebyside>, and in any case may have a specified additional blank working space of specified height. Page breaks can be specified, and the four margins on a page can be independently controlled. So if you want to create ancillary worksheets for your project, and you like to use *all* of the space on a printed page, then there is some layout control to support that.

Notice that all of this layout control is an exception to the philosophy of PreTeXt. So in particular, margins and working space do not appear in the HTML output. We do give a visual indication where a page break in a <worksheet> is placed. An author might wish to collect all of the worksheets in a book, for printing as an "activity book", and so there are plans (2018-08-11) to support and automate that process. Details for authoring worksheets can be found in Subsection 4.7.3.

3.12 References

Like <exercises>, a <references> division may go anywhere a more typical division could go. This allows for things like a "Further Reading" list at the end of every chapter of a book. These are populated with <biblio> items that are individual bibliographic entries. Support is presently very minimal, but is planned to improve.

3.13 Figures, Tables, Listings, Lists

Some elements in PreTeXt are **containers**, meaning they do not stand by themselves, but are meant to be filled with other elements. Other elements are **atomic**, meaning they cannot be decomposed into smaller elements. A canonical example is the case of a <figure>, which is a container meant to be filled with other elements. One such element is an <image>, which is atomic. Note that the figure can be filled with other items, and that an image may appear inside other elements, including as a child of a division, perhaps as a peer within a run of paragraphs. A purpose of the container is typically to provide a number and some text (a title, caption, or similar) as identification or description, in addition to indicating to conversions the necessity for some visual formatting (such as small amounts of separating vertical space above and below). Note that an author provides the caption, while PreTeXt provides the vertical spacing.

Here we provide very brief descriptions of these four containers. Be sure to consult the in-depth topics for more details on specifics. Generally, the common thread here is that these containers contain text or graphic elements that has a two-dimensional quality to it. To different degrees the content is rigid. Unlike a paragraph, which could be unwound into a single (linear) long sequence of characters, *something* about the contents of these containers would be lost if stretched out in one dimension. To reflect this rigid two-dimensional flavor, we refer to these objects (and their containers) as **planar**.

A <figure> is the most general planar container. It can hold an <image>, a <audio>, a <video>, and more. A <caption> is authored early as metadata, but will likely render below. A <title> can be used for cross-references or in lists of figures, but may not render where authored. See Section 4.17 for full details.

A <table> is the container for a <tabular>, an atomic element which is the only allowed content of a table. A <title> is the identifying information, and renders above the rows and columns of the table. A variety of notes (to appear below the table) are possible (not implemented as of 2021-12-10). The elements we provide to describe a table are heavily influenced by the discussion in *Chicago Manual of Style* [1, Chapter 13], which is worth reading if your project has many important tables. See Section 4.18 for full details.

A <listing> is a container for computer code or programs, to support projects in computer science and other technical disciplines. These languages often rely on indentation (Python) or even exact column numbers of text (FORTRAN), hence a planar quality. Other languages can be written syntactically-correct as one long reflowable line (C, Pascal), but are impractical to do so as part of an exposition. So the allowed content is a <program> or <console>, which will respect indentation, use monospace fonts, and include syntax highlighting. Otherwise, a listing is very similar to a figure in how a caption and title are handled. See Section 4.19 for full details.

A <list> may not be what you think it is. An actual list (be it ordered, unordered, or description) is a common and popular device for organizing information. Start at Section 3.8 for details on lists. Since these lists are considered part of a sentence (within a paragraph), or a part of a paragraph, it is hard to create a cross-reference to them. So when you have a list that is important to mention elsewhere, you can create a **named list** with the `<list>` element, a container that has an optional `<introduction>`, followed by a actual list, and then an optional `<conclusion>`. An example might be a laboratory procedure, such as the steps necessary to dissect a frog. Like a `<table>` this element should have a `<title>`, and will be given a number. Using a new line for each new list item, and conveying nesting with indentation gives a list a planar quality. See Section 4.20 for full details.

We use a `<table>` to summarize these similar containers.

Table 3.13.1 Containers for Planar Content

	Figure	Table	Listing	List
Element	`figure`	`table`	`listing`	`list`
Contents	various	`tabular`	`program`, or `console`	`introduction` `ol`, `ul`, or `dl` `conclusion`
Title	×	×	×	×
Caption	×		×	
Notes		×		
Details	Section 4.17	Section 4.18	Section 4.19	Section 4.20

3.14 Exceptional Characters

An advantage of XML syntax is that very few characters are reserved for the language's use, and thus very few characters need to be escaped. Of course, there is always the need to escape the escape character.

The escape character for XML is the ampersand, &. The other dangerous character is the left angle bracket, the "less than," <. If you like to be symmetric, you can also handle the right angle bracket, the "greater than," >, similarly. Single and double quotation marks are used to delimit attributes, so are part of the XML specification, but do not present difficulties in narrative text.

These first two characters are interpreted by the XML processor very early in the analysis of your source. So they need to be authored specially via the **XML entities** `&amp;` and `&lt;`. In practice, escaping > is rarely necessary. So fundamentally within PreTeXt there are just two characters to type carefully or exceptionally.

If you consistently follow the prescription in the previous paragraph you will avoid a descent into escape-character hell and avoid a lot of head-scratching. In particular, you should have no need of the `<![CDATA[ ]]>` mechanism of XML, so *please* just forget we even mentioned it. But see Subsubsection 4.1.4.2 if you are curious, or want a more thorough discussion.

3.15 Nontrivial Characters

Quality typography is more expansive than the limited capabilities of computer keyboards, whose history is rooted in the machines known as typewriters. Some characters are never present on a keyboard (e.g., a pilcrow), while others are ambiguous. Is a diagonal line a slash used to separate information (either/or), or is it a solidus used to form a simple fraction such as ¾? Table 3.15.1 is a *sample* of how PreTeXt addresses this. (The last two are the technical exceptions imposed by using a markup language, see Section 3.14.) See Section 4.1 for a more comprehensive and detailed discussion.

Table 3.15.1 Nontrivial Characters

To get this:	Type this:	To get this:	Type this
…	<ellipsis/>	________	<fillin/>
⚙	<icon name="gear"/>	🔧	<icon name="wrench"/>
PrtScn	<kbd>PrtScn</kbd>	⇧	<kbd name="shift"/>
–	<ndash/>	—	<mdash/>
®	<registered/>	©	<copyright/>
‰	<permille/>	¶	<pilcrow/>
§	<section-mark/>	·	<midpoint/>
™	<trademark/>	⁓	<swungdash/>
×	<times/>	±	<plusminus/>
⁄	<solidus/>	÷	<obelus/>
°	<degree/>		
′	<prime/>	″	<dblprime/>
&	&amp;	<	&lt;

3.16 Verbatim and Literal Text

Typesetting literal text, usually in a monospace font, can sometimes be tricky. For short bits of such text, as part of a sentence in a paragraph, or in a caption or title, use the <c> tag, which is short for "code." For much longer blocks of literal text, with line breaks that are to be preserved, use the <cd> element within a paragraph ("code display"). Outside a paragraph, most anywhere you could place a regular paragraph, use the <pre> tag, which is short for "pre-formatted".

For the content of a <pre> element, the indentation will be preserved, though an equal amount of leading white-space will be stripped from every line, so as to keep the code shifted left as far as possible.

The behavior of these two tags is to preserve characters exactly. Certainly the ASCII character set will behave as expected, and Unicode characters will migrate successfully to output formats based on HTML. As mentioned in Section 3.14 the ampersand and left angle bracket will confuse the initial XML processing. So use the XML entities &amp; and &lt; to represent these characters to the PreTeXt conversion tool, the PreTeXt-CLI. See Section 4.4 for further details.

3.17 Sage

Sage[1] is an open source library of computational routines for symbolic, exact and numerical mathematics. It is designed to be a "viable free open source alternative to Magma, Maple, Mathematica, and Matlab." PreTeXt contains extensive support for including example Sage into your document.

A typical use of the <sage> tag is to include an <input> element, followed by an <output> element. The content of the <input> element may be presented statically in PDF output, or dynamically as a Sage Cell in an output format based on HTML. Of course, for output as a CoCalc or Jupyter worksheet, the Sage code is presented in the worksheet's native format.

The content of the <output> element is included in PDF output, but not in dynamic instances, since it can be re-computed. Notably, there is a conversion which pairs input and output into a single file in the format used by Sage's doctest framework. So if expected output is provided, it becomes automatic to identify when Sage has diverged from your expectations, and you can adjust your examples accordingly.

The Sage Cell Server can also be configured to interpret different languages, because Sage by default contains everything needed to evaluate code in these languages. This is done by providing a @language attribute, where possible values are sage, gap, gp, html, macaulay2, maxima, octave, python, r, and singular. The default is sage.

Note that the dynamic formats (including the Sage Cell) may run Sage "interacts," so that is possible to embed interactive demonstrations into your dynamic output formats.

[1] www.sagemath.org/

3.18 Interactives

We strive to make it simple for authors to incorporate interactive demonstrations in the online output of their projects. Of course, this prompts the question of what to do with this content in less-capable formats like PDF or braille.

The <interactive> element provides a consistent way to specify these demonstrations. There are many possibilities, but perhaps they can be grouped mostly into three broad classes.

Server Hosted Interactive demonstrations hosted at external sites, such as those from GeoGebra or Desmos, can be included simply by providing the appropriate identifying information, much like the way you would specify a YouTube video (Subsection ??).

Source Code Some interactives can be described by source code that you include in your PreTeXt source. Examples include JessieCode, CircuitJS, and Sage Interacts. We also support GeoGebra this way. This is similar to how we employ Sage Cells, but without as much specialization.

Roll Your Own If you know HTML, CSS, and Javascript, you can provide your own routines and libraries to incorporate any sort of demonstration you can imagine and can code.

See Section 4.22 for details.

For output formats where executing an <interactive> would be impossible, we manufacture a static version. This includes a screenshot of the demonstration (automatically generated, or supplied by the author) and a Quick Response (QR) code that will point to a **standalone** HTML page that contains the interactive. Again, see Section 4.22 for details.

3.19 Side-by-Side Panels

A <sidebyside> is a useful organization of elements in a horizontal layout, and so begins to blur the line between content and presentation. While we default to organizing information in a vertical sequence, it is often desirable to organize smaller elements adjacent to each other horizontally. Specifically, images, tabular, figures, tables, paragraphs, and more, may all be combined and there is good control over vertical and horizontal alignment. Captioning, both overall and individually, is especially flexible. An <sbsgroup> ("side-by-side group") collects multiple <sidebyside> to stack vertically, which allows for displays in grids. See Section 4.23 for details.

3.20 Mathematical Results

Definitions, theorems, corollaries, etc. are supported by the tags: <theorem>, <corollary>, <lemma>, <algorithm>, <proposition>, <claim>, <fact>, <definition>, <conjecture>, <axiom>, and <principle>. Each may have a <title> (strongly encouraged), and then contains a <statement> which is a sequence of paragraphs and other elements. As appropriate, some of these elements (such as a <lemma>) may contain an optional <proof> (or several), while other elements may not have a <proof> (such as a <conjecture>).

A <definition> is a natural place to define notation as well (see Section 3.23), and to use the <term> tag to identify the terminology being defined.

In order to assist readers locating numbered items, these items are all numbered consecutively in a group that includes <example>s, <remark>s and inline <exercise>s.

3.21 Front Matter

In the beginning of your <book> or <article> you can have a <frontmatter> element that contains various items that would precede your first <chapter> or <section> (respectively). Possibilities include <titlepage>, <colophon>, <biography>, <abstract>, <dedication>, <acknowledgement>, <foreword>, and <preface>. Some of these may be duplicated (e.g., several prefaces for multiple editions), many of these items are restricted to books (e.g., a foreword), and some items are restricted to articles (e.g., an abstract). The schema (Chapter ??) will help you place them in the right order in your source. See Section 4.24 for details.

The `<frontmatter>` is also employed in other types of documents, such as a `<slideshow>`, in similar, but not identical, ways.

3.22 Back Matter

Similar to front matter, there is material you might wish to include after your book's final `<chapter>` or your article's final `<section>`. Possibilities to place in a `<backmatter>` include `<appendix>`, `<references>`, `<glossary>`, `<solutions>`, `<index>`, and `<colophon>`. There are empty tags you can place into an appendix to generate lists of notation, or lists of particular elements of your choice, such as a list of figures. A similar empty element actually generates the index, `<index-list>`, but you will almost always want to place it into the `<index>` division. See Section 4.25 for details on the back matter generally. Also, be sure to read about the powerful and flexible automatic list feature (Section 4.28), which is not restricted to just the back matter.

3.23 Index and Notation Entries

Construction of an index and a list of notation is accomplished by placing information into your text in the appropriate places in the right way.

The `<idx>` tags denote an index entry. These should be placed within the element that they describe. By this we mean that an `<idx>` element can be placed within a `<theorem>` to refer to just that theorem, or it might be placed within a `<subsection>` to refer to that subsection. When you do this, a natural place to place the `<idx>` is right after the `<title>` and similar metadata. In this way, electronic versions of your work can have an index that is more informative than a traditional index that uses just page numbers, since it will be apparent while reading and using the index just what type of object the entry refers to. (See the end of this Guide in an electronic format for an example, Index). Note that the text contained within the `<idx>` tags does not actually appear in the article—it only serves to mark the location the index entry points to. You can have several levels of headings by structuring your `<idx>` element with up to three `<h>` tags ("h" for "heading"). Additionally, you can use `<see>` or `<seealso>` for cross-references within the index.

See Section 4.26 for more details and the finer points of creating index entries.

A similar device is used to create a list of notation for a technical (mathematical) work. Place a `<notation>` element as close as possible to the place where notation is first introduced. If you use the `<definition>` tag for your definitions, then this is a very natural place to also introduce notation. Inside of `<notation>` use the `<usage>` tag to include a short example of the notation in use, wrapped in an `<m>` tag and use LaTeX syntax (as usual). The `<description>` tag should contain a very short description in words of what the notation is for. So "center of a group" would be a good description to accompany the usage "$Z(G)$."

See Section 4.27 for more details and the finer points of creating notation entries.

3.24 WeBWorK Exercises

It is possible to author WeBWorK automated homework problems directly within your source. A static version will be rendered in your LaTeX/PDF/print output, and a "live" version will be rendered into HTML output (though a student is not able to authenticate against a course). However, you can also extract *all* of the WeBWorK questions from a textbook into a single archive suitable for uploading into a traditional WeBWorK server. This is a big topic, so see the dedicated Chapter **??** for details.

3.25 URLs and External References

The `<url>` tag always requires an `@href` attribute. Usually this will be a complete address for some external web page, or other external resource. The `@@visual` attribute is sometimes mandatory, but sometimes optional. It should provide a simplified version of the URL for use in print, or similar situations. Finally, you can provide content for the `<url>` element, which will become the clickable text in most realizations.

If the `<url>` element is empty (no content), then the value of the `@href` attribute or the optional `@visual` attribute will be the link text, with a preference for the latter. When you instead provide content, you can use PreTeXt elements much like any other piece of text that would occur in a paragraph. In this case, a `@visual` attribute is now highly recommended, as an alternative to the content, providing information about the actual URL for non-electronic formats like print. A default version of the value of the `@href` attribute will be used in its absence. This visual version of the URL will appear in a footnote.

See Section 4.29 for an example and full details. There is a similar `<dataurl>` element for pointing to supporting files, see Subsection 4.29.2.

3.26 Video

Videos, either author-hosted, via a URL, or hosted on YouTube, may be embedded in a PreTeXt document that is converted to HTML, and may be optionally "popped-out" to view on another page. For a YouTube video, it is simplicity itself, as an author need only supply the identification string, and all the details of the embedding are handled by PreTeXt. See Section 4.30 for details.

3.27 Scientific Units

If you are writing about science or engineering, or even if you are not, there is extensive support for scientific units. So, for example, you could author a force in metric units as

```
<quantity>
    <mag>20.7</mag>
    <unit prefix="kilo" base="gram" />
    <unit base="meter" />
    <per  base="second" exp="2" />
</quantity>
```

This would be rendered as $20.7\,\frac{\mathrm{kg}\cdot\mathrm{m}}{\mathrm{s}^2}$. More in Section **??**.

3.28 Localization

We are interested in helping authors produce documents with open licenses around the entirety of the world. PreTeXt provides infrastructure for doing so (or **internationalization**) in accordance with guidelines of the W3C Internationalization (I18n) Activity[1] at W3C (`www.w3.org`).

In order to actually adapt PreTeXt to another language (**localization**), there are two requirements:

- A file for localization into the desired locale.
- The author adds an `xml:lang` attribute on the `<pretext>` element.

See Section **??** for more details.

3.29 Accessibility

The Web Accessibility Initiative[1] at W3C (`www.w3.org`) says:

> The Web is fundamentally designed to work for all people, whatever their hardware, software, language, culture, location, or physical or mental ability. When the Web meets this goal, it is accessible to people with a diverse range of hearing, movement, sight, and cognitive ability.

[1] `www.w3.org/International/`
[1] `www.w3.org/standards/webdesign/accessibility`

> Thus the impact of disability is radically changed on the Web because the Web removes barriers to communication and interaction that many people face in the physical world. However, when websites, web technologies, or web tools are badly designed, they can create barriers that exclude people from using the Web.

Since we are interested in helping authors produce documents with open licenses, and we concentrate on employing open standards for the HTML output we create, we are ideally positioned to help you easily create highly-accessible documents. There are many technical features which happen automatically, and there are some features which we make available for your use as an author and which only an author can provide, or elect to use. Before getting too deep into your project, review Section **??** for full details and ways you can make the HTML version of your document more accessible, and more useful for a wider audience.

3.30 Slides

In addition to articles and books, support for authoring slideshows is also available, but is experimental (so tags and attributes are subject to change).

Currently, support for producing Reveal.js[1] and Beamer LaTeX[2] slideshows is available. Details on authoring may be found in Section **??**, and publishing details are in Chapter **??**.

Also, future support for annotating PreTeXt books and articles to export PreTeXt slides is in the works, but not yet available.

3.31 Literate Programming

If you know what **literate programming** is, then you may not be surprised to learn that PreTeXt provides excellent support primarily with two additional elements: `<fragment>` and `<fragref>`.

If you have never heard of literate programming, it is a way to mix code and documentation for a computer program in a single file. But rather than typographically simple code comments, you have the full power of every other feature in PreTeXt and all of the possible output formats. Furthermore, you can arrange your code in an order that might make more sense to a human (top-down, bottom-up, a mixture, or ...) and PreTeXt will rearrange the code into an order that the compiler or interpreter understands.

So the idea is that from one file you get a program for the compiler, and a beautiful, typeset explanation for a human reader in any format PreTeXt supports. An accessible introduction is Knuth's description of his `WEB` system for the Pascal programming language [2], or many more resources are at the literate programming site[1]. Full details on the PreTeXt implementation are at Section **??**.

[1] `revealjs.com`
[2] `en.wikipedia.org/wiki/Beamer_(LaTeX)`
[1] `www.literateprogramming.com`

Chapter 4

Topics

This long chapter provides the main documentation for a variety of the features of PreTeXt. Some sections are just stubs and need to be written. Requests for sections to prioritize are welcome, though some sections are waiting for features to stabilize.

4.1 Paragraphs

Much of your writing will happen in paragraphs, delimited by the simple tag, `<p>`. You are reading one right now. They are a basic building block of divisions, and also a basic building block of other structures. For example, an ordered list, `<ol>`, contains a sequence of list items, `<li>`, and a typical list item might be a sequence of paragraphs. (Do not confuse this element with the anomalous `<paragraphs>` subdivision, Section 4.6).

Paragraphs are a choke-point of sorts. Many tags can *only* be used within paragraphs, and many others *cannot* be used within paragraphs. Notice too, that we do a certain amount of manipulation of whitespace in a paragraph, in ways that you may not even notice.

The following subsections together contain allowed, or encouraged, markup within a paragraph. Many of these may be used in captions and titles, but some of the more complicated constructions (which appear later here) cannot be used in captions or titles.

One more comment: typewriters, computer keyboards, and the ASCII character set have together conspired to limit the full range of characters that typographers and printers have used historically. A case in point is the hyphen, which is a single key on a keyboard. However, there are at least three common dashes of differing lengths (hyphen, en dash, and em dash), and in the context of mathematics or a computer program, the hyphen might be the binary operation of subtraction or the unary operation of negation. Another example is the "upright", or "dumb", quote mark that is a single key on a keyboard, while careful typography will employ "smart", or "curly", quote marks that have left and right variants. (Sometimes called "66" and "99" based on their shape.) PreTeXt will help you navigate this complexity, but you will want to use keyboard characters or markup appropriately. So if you care about communicating clearly, and making your writing easy for a reader to use, absorb the details that follow and the philosophy they implement.

We will say it again. PreTeXt is a markup language, and our various output formats (LaTeX, HTML, EPUB, Jupyter notebooks) in turn employ markup languages. These use different escape characters and give different characters special meanings. Our job is to insulate you from this variety, so you can concentrate on authoring your *ideas*.

We begin with some simple "grouping" elements which contain several excellent examples of the importance and utility of careful markup. There is a plethora of empty tags for individual characters, and these are very important (see Subsection 4.1.4). We defer them to the end of this section, since they are not as instructive, but do not think this means they are an afterthought. They can be extremely critical for successful conversions. Also do not miss Best Practice 4.1.8 in the conclusion of this section.

4.1.1 Simple Markup in Paragraphs

Beyond empty tags that translate to various characters, there are relatively simple tags that can call attention to various *portions* of a sentence, or generate more complicated constructions than described above.

Most, if not all, of the markup in this subsection may also be used within titles and captions, though they might lose some of their features when used in a title, especially when the title is duplicated in other contexts, such as a Table of Contents.

`<q>`, quotes, “group”. This is the first of several grouping tags, using characters with left and right variants, and some of the most common markup in your writing. Presentation uses double quote marks that are **smart quotes**, meaning that they look different in their opening and closing variants. (See `<blockquote>` for extensive runs of quoted text that can stand alone, and which can carry an attribution.)

`<sq>`, single quotes, ‘group’. Perhaps less-often used than `<q>`, so a couple more characters to type. Presentation is paired single-quotes, opening and closing.

`<angles>`, angle brackets, ⟨group⟩. Left and right angle brackets to enclose a phrase. This is not for creating a set of generators in mathematics, use the appropriate mathematics tag and syntax for that. Note also that the characters used here are definitely distinct from the inequality symbols, < and >.

`<dblbrackets>`, double square brackets, ⟦group⟧. Double left and right square brackets to enclose a phrase. This is not for grouping expressions in mathematics, use the appropriate mathematics tag and syntax for that. These are used in the analysis of texts to note various restorations or deletions. Inquire if you feel there should be more semantic markup for this purpose.

`<em>`, emphasis, *important*. Use this element to surround characters in a phrase that is to be emphasized. This will typically be rendered in italics, though this choice is left to the implementation of a particular conversion. See also, `<alert>`.

If you are new to using a markup language, this is a place to stop and think. As a PreTeXt author you are never able to say, “I want this text to appear in italics.” Rather, you specify that certain text has a certain purpose or meaning. Emphasis is a way of *calling attention* to a portion of a sentence or paragraph. A font change (to italic) is a common and effective device. But a particular format might have a better, or different, way to achieve this. Perhaps in an electronic format, the letters are animated and dance up and down. (Just kidding. But you may be reminded of frequent blinking text in the early days of web design, supported by a non-standard `<blink>` element.) Seriously, now would be a good time to review Section 1.1.

`<alert>`, alert, *critical*. Use this to heavily emphasize something to a greater degree than just emphasis. Maybe think of it as ***SHOUTING***. Bold italic font, or a bright color, or both, would be normal choices for presentation. Overuse of this tag will dilute its effectiveness.

`<term>`, terminology, larvae. Use this to identify a word or phrase that is being defined, in contrast to actually using a structured `<definition>`. Typical presentation is a bold font. Caution: the use of this tag is to communicate a defined term and converters may make use of this interpretation, given the importance of definitions in scholarly work. It would be considered **tag abuse** to use this tag to simply bold a word or phrase for some other reason, perhaps as an alternative to `<em>` or `<alert>`.

`<foreign>`, foreign words, idioms, phrases, *Hola*. This tag is used to identify words or phrases from a language other than the main one used for the overall document. It is best practice to use a `@xml:lang` attribute to identify the language, since this will assist screen readers and hyphenation algorithms. We may also recognize the need for a different script (font). Usual presentation is italics for languages using a Latin script. This should not be used for entire paragraphs as a way of assisting with a translation of an entire document.

Note that when we use italics for emphasis *and* to point out foreign words or phrases, there is a loss of information in our output. In other words, we can no longer reliably (in an automated way) convert our output back to equivalent PreTeXt source from its visual representations. *C’est la vie.* See Section 1.1 again.

`<pubtitle>`, `<articletitle>`, titles of books and articles. These provide the ability to typographically distinguish the title of another work, and are not a replacement for careful bibliographies and citations. Use `<pubtitle>` for longer, complete works, such as books, plays, or entire websites. Use `<articletitle>` for shorter, component works, such as a chapter of a book, a research article, or a single webpage.

Presentation for a longer work will be italics or an oblique (slanted) font, and for a shorter work, the title will simply be quoted.

`<abbr>`, `<init>`, `<acro>`, abbreviation, initialism, acronym, Mr., XML, SCUBA. An abbreviation is a condensed or shortened version of some word or phrase, such as MR. for "Mister." Converters should take care with periods (full stop) inside an `<abbr>` as distinguished from the end of a sentence (which may not always be clear given the absence of a tag delimiting sentences). An initialism is an abbreviation read as a sequence of letters, often the first letter of words in a phrase, such as HTML for "HyperText Markup Language." An acronym is much like an initialism, but the letters are read as a pronounceable word (which sometimes actually enters the language as a word, such as "radar" which began as RAdio Detection And Ranging). An example is SCUBA which stands for "Self-Contained Underwater Breathing Apparatus." Initialisms and acronyms may be presented in a small-capitals font or as regular capitals reduced in size.

`<delete>`, `<insert>`, `<stale>`, editing assistance, ~~gone~~, new, ~~old~~. These denote portions of a text that is being changed in some way, presumably as part of an editing process. Conceivably, they could be managed by some other tool acting on your source. Stale text is that which is slated for removal eventually, but is left in place so that it may be consulted. There is no support presently for actually deleting or incorporating text, though that would be a reasonable feature request.

Red and green, for leaving and entering, are natural choices for presentation. But in consideration of those readers who cannot always distinguish different colors, other devices, such as strikethrough or underlining, should also be employed.

`<tag>`, `<tage>`, `<attr>`, tag, empty tag, attribute, `<section>`, `<hash/>`, `@width`. These are PreTeXt tags for when we write about PreTeXt and need to discuss tags, empty tags, and attributes. Given how we design PreTeXt tags the content of these elements should only be the 26 lower-case letters and a dash/hyphen. These should render in ways that make the three types of language elements obvious without much further discussion. Just a bit self-serving, but not unjustified.

`<taxon>`, scientific names, *Escherichia coli*. This element may surround a full scientific name, resulting in presentation in italics. There are subelements `<genus>` and `<species>` which may be used to delineate those components.

A `@ncbi` attribute on `<taxon>` accepts an identifier from the National Center for Biotechnology Information[1]. Feature requests for ways to make this more useful are welcome.

`<fn>`, footnotes. A footnote can be inserted in a paragraph and a mark will be left behind. Where the content of the footnote goes depends on the capabilities of the output format. Because a footnote allows you to begin a new piece of text *anywhere*, it can be difficult to handle technically. For this reason it is banned from places like titles and its possible content is limited (for openers, no paragraphs).

A footnote is the farthest thing from structured writing that we can think of. It can go anywhere. Resist the temptation to use it, and your writing will improve. We frequently entertain the thought of making footnotes impossible in PreTeXt. See the `<aside>` element for a possible alternative.

`<m>`, mathematics, $x^2 + y^2$. Simple, inline expressions using mathematical notation may be used in paragraphs, and in titles and captions. The syntax is LaTeX. See Section 4.9 for full details.

`<c>`, code, verbatim text, literal text, `import`. Short bursts of raw, or verbatim, text can be wrapped in the `<c>` element. Strictly speaking, "code" is a misnomer, as the text may be anything you need to communicate exactly as one would type it at a keyboard or as input to a computer program. Anything longer than a handful of characters,

[1] `www.ncbi.nlm.nih.gov`

or needing accurate line breaks should consider the `<cd>`, `<pre>`, `<program>` or `<console>` tags. Presentation is normally a monospaced sans serif font, perhaps of a slightly heavier weight, and designed for the job with features such as unambiguous zeros (versus the letter ‘oh’). See Section 4.4 for details.

`<email>`, email address, `nobody@example.com`. Very similar to the `<c>` tag, this may be used to get a monospace presentation of an email address, possibly as an active link in some formats.

4.1.2 Cross-References and Paragraphs

There are several devices for creating cross-references. Generally, these are unwise (or banned) in titles, and if allowed may be inactive in certain portions of an electronic output format (such as when migrating to a Table of Contents).

`<url>`, linking external resources. This is a cross-reference *to* some item separate and distinct for your document.
A Uniform Resource Locator (URL) is, loosely speaking, an Internet address for some item. Presentation depends on the capabilities of the output format to serve the resource. There is a mandatory attribute, `@href`, that contains the full URL, including a protocol (such as `http://`). Used as an empty tag, the visual text will be the exact contents of the `@href` attribute. So, `http://www.example.com` can be achieved with

```
<url href="http://www.example.com"/>
```

You may also wish to provide some text other than the actual URL, which you can specify as the content of the `<url>` element. For example, IANA Test Site[2] can be achieved with

```
<url href="http://www.example.com">IANA Test Site</url>
```

In order to have a URL occur in print output in a useful way, and in electronic output in an active way, the `@visual` attribute can be used to display the visual portion as verbatim text in a footnote. So illustrating again, we get `example.com` from

```
<url href="http://www.example.com" visual="example.com"/>
```

Notice the necessity and/or desirability of marking the text in a way that distinguishes it as literal text.
Note also that this tag is meant for *external* resources, so see the `<xref>` element (below) or Section 4.5 for ways to link internally (i.e. within your document).

`<xref>`, cross-references. This is a cross-reference *to* some item contained within your document.
Extensive and intuitive capabilities for cross-referencing are a primary feature of PreTeXt. Typical use is an empty tag with the `@ref` attribute specifying the value of an `@xml:id` on the **target** of the cross-reference. This should work easily without much more instruction, but familiarize yourself with the details in Section 4.5 to get the most out of some the available options.

`<idx>`, index target. This indicates that the containing structure (theorem, example, etc.), or top-level paragraph, should be the *target* of an entry of the index (a special sort of cross-reference). See Section 3.23 and Section 4.26 for general details.

`<notation>`, index target. This indicates that the containing definition, or top-level paragraph, should be the *target* of an entry of the list of notation (a special sort of cross-reference). See Section 3.23 and Section 4.27 for general details.

4.1.3 Structured Markup in Paragraphs

There are three categories of items which typically are structured further, and which are almost entirely restricted to appearing in a paragraph. Given their complexity, details are covered in other sections of this guide.

[2] `www.example.com`

Lists. With only one major exception (and three minor ones), a list *must* appear within a paragraph. See Section 3.8 for an introduction and Section 4.11 for precise details.

Display Mathematics. Displayed mathematics, which is a single equation or a sequence of (aligned) equations, can only be placed within a paragraph. The relevant tags are `<me>`, `<men>`, `<md>`, and `<mdn>`, with the latter two necessarily structured with `<mrow>` elements. See Section 3.6 for an introduction and Section 4.9 for precise details.

Display Verbatim. The `<cd>` tag, by analogy with the `<md>` tag for displayed mathematics, may be used to display one or more lines of verbatim text (such as a series of commands), possibly structured with the `<cline>` tag. See Section 3.16 for an introduction and Section 4.4 for precise details.

This should not be confused with the `<pre>`, `<console>`, or `<program>` tags, which have slightly different uses, and all of which must be used *outside* of a paragraph.

4.1.4 Characters in Paragraphs

Some keyboard characters are unambiguous, for example, the percent sign, %. Other keyboard characters are poor replacements for several different characters. Is a slash, /, being used to separate information/ideas, or is it a **solidus** being used to form a fraction such as ¾? Other characters, such as per-mille, ‰, are not present on keyboards at all. We organize this section according to these types of distinctions.

4.1.4.1 Unambiguous Keyboard Characters

The keyboard characters `, ~, !, @, #, $, %, ^, *, (,), _, =, +, [,], {, }, \, |, :, ;, and , are entered as-is and are only rendered one way. Easy.

Of course, the fifty-two Latin letters, and ten decimal digits, are also in this category. If you have an international, or bilingual, or country-specific keyboard, then common accented versions of Latin letters (as used in Europe and the Western Hemisphere) may also be used directly from your keyboard.

4.1.4.2 Exceptional Keyboard Characters

XML is a **markup language**, which in part means that some keyboard characters are co-opted to signal the start of markup. For XML this character is the less-than symbol, <. It signals the start of a **tag**, and then an opening tag ends with a greater-than symbol, >, while a closing tag has an extra / right after the <.

This begs the question: if a < is used in our XML source to signal the start of a tag, then how did we get one to appear here in this sentence without mistakenly starting a tag? Once a markup language gives some characters special meanings, then there needs to be an **escape character**. For XML the escape character is the ampersand, &. So to author the < and > symbols, we type **escaped** versions: `&lt;` and `&gt;`.

I hear you now say, "But now we just took the & out of the running and gave it a special meaning. How do we get an ampersand?" Easy, use the escaped version: `&amp;`.

So the short answer is: never, ever type the < or & keyboard characters in isolation. The very beginning of the processing of XML (i.e. PreTeXt) will fail fatally on these characters. Instead, always use the sequences `&lt;` and `&amp;` and then very early the XML processing will convert them to characters, *without* interpreting them as signals for aspects of the markup.

It does not seem necessary to author > as `&gt;`, though there is no real harm in doing so. The two other characters with escaped versions are the single and double quotes, `'` and `"`, which have escaped version of `&apos;` and `&quot;` (respectively). These are only necessary for attribute values, and we have been careful to design PreTeXt so that they are not necessary.

Remark 4.1.1 Excessive Escaped Characters. If you know another markup language, such as TeX, LaTeX, Markdown, JSON, or PGML, think about how many characters have been given special meanings, and the subsequent necessity to use escaped versions. And if you want to write about computer languages, realize that each such language also gives certain keyboard characters special meanings.

XML only has five exceptional characters, and in your daily use, PreTeXt really only requires you to be aware of two, the minimum necessary for a markup language.

Best Practice 4.1.2 A CDATA Section is Never Necessary. We hate to mention it, but sooner or later, we need to have an uncomfortable discussion about the misunderstood CDATA section, and risk confusing the rest of this subsubsection. And this is the place. But you can come back later, if you wish.

You will read other places about very special markup known as a CDATA section. The name stands for **character data**, which means "all characters, no markup". Think of it as switching off the XML processing for a while, so in particular, `&`, `<`, `>` no longer have any special meaning at all. That *could* be nice, but realize that now there is no opportunity to have any markup present using XML syntax, since it is ineffective.

A CDATA section is always a convenience and is never necessary.

When would it be convenient? Maybe you have some LaTeX inside an `<md>` with a large matrix that uses lots of ampersands to separate the entries. Inside a CDATA you can author it with bare & rather than a plethora of `&amp;` or `\amp`. But you lose the ability to include an `<xref>` in that CDATA, so you need to be surgical about its scope. Perhaps a Tikz diagram in a `<latex-image>` has a multitude of `<->` or a chunk of Sage code in an `<input>` has a lot of finitely-generated algebraic structures authored as `R.<x> = ...` (which is not even legal Python syntax either!). These places where there is little, or no, markup could be *convenient* places to use a CDATA. Be sure to read the warning at Item 7 in Section **??** before you go all-in.

4.1.4.3 Ambiguous Keyboard Characters

Some keyboard characters have a primary interpretation, and are imitations of other typographic characters. Your output will be of higher quality if you understand these distinctions and employ the proper variant.

Table 4.1.3 Ambiguous Keyboard Characters and Alternatives

Keyboard	Primary	Notes
`/`	(forward) slash	`<solidus/>` is a fraction bar,⁄
`'`	apostrophe	`<rsq/>` is a right single quote, '
`` ` ``	backtick	`<lsq/>` is a right single quote, `
`.`	period	abbreviations *and* end-of-sentence
`-`	hyphen	See dashes, and arithmetic
`"`	upright double quote	`<lq/>` is ``, `<rq/>` is ''

Note that the four quote marks (left/right, single/double) are meant for the actual characters. Always use the grouping constructions described above (i.e. `<q>` and `<sq>`) when grouping a phrase with quote marks. Note, too, that there is never a good reason to use the keyboard quote character (") unless you are creating some sort of verbatim text, such as a program listing or describing literal keyboard input.

When creating print or PDF via LaTeX a period may get different trailing space depending on location and context, generally being its use in abbreviations or to conclude a sentence. We do not yet have this dual-use under control.

4.1.4.4 Extraordinary Characters

Some characters or symbols are typically not available on a keyboard, so we provide empty elements. Many of these may be entered directly into your source as Unicode characters, and they will do well in your HTML output. However, these may fail entirely if you create print or PDF via LaTeX using the `pdflatex` engine. Furthermore, even for HTML output there may be several Unicode characters that are very similar.

So again, for the best quality output be aware of these elements and use them. Please suggest additions if you do not find what you need and are resorting to Unicode characters.

`<ellipsis/>`, …, ellipsis. Typically three low dots with no intervening space, to indicate a continuation. This will always perform better than three consecutive periods.

`<midpoint/>`, ·, midpoint. A small centered (vertically) dot, which can be used to separate pieces of information, especially in displayed text (i.e. outside of paragraphs). Not to be confused with a bullet preceding a list item, or multiplication in mathematics.

`<swungdash/>`, ⁓, swung dash. Another decorative separator, not to be confused with the keyboard tilde character since it is wider and thicker.

`<permille/>`, ‰, per mille. Like per cent, but now a number expressed as its product with 1000 (rather than with 100).

`<pilcrow/>`, ¶, pilcrow, paragraph mark. Mark used historically to indicate the start of an internal paragraph, and in a more modern use, to indicate a permalink.

`<section-mark/>`, §, section mark. Used to prefix the number of a section, or other division. (So the word section is being used generically here.)

`<copyright/>`, ©, copyright. The symbol used in publishing, legal, or business contexts. For a PreTeXt project, copyright information can be specified within the `<colophon>` portion of the `<frontmatter>`.

`<trademark/>`, ™, trademark. The symbol used in legal or business contexts.

`<registered/>`, ®, registered. The symbol used in legal or business contexts.

Table 4.1.4 Extraordinary Characters and Their Empty Elements

Character	Name	Element
…	ellipsis	`<ellipsis/>`
·	midpoint	`<midpoint/>`
⁓	swung dash	`<swungdash/>`
‰	per-mille	`<permille/>`
¶	pilcrow	`<pilcrow/>`
§	section-mark	`<section-mark/>`
©	copyright	`<copyright/>`
™	trademark	`<trademark/>`
®	registered	`<registered/>`

4.1.4.5 Accented Characters

The second 128 Unicode characters (hex `80` to `FF`) contain many of the most frequently-used accented characters in Western languages, along with niceties such as the German *eszett*, ß, or the Scandinavian *æsc*, æ, an a-e ligature. Like the fifty-two Latin letters (part of the *first* 128 Unicode characters), these may be used as-is. They may be present on your keyboard, or you may need to learn keyboard shortcuts or specifics of your operating system to enter them as Unicode characters. In a pinch, you can often cut-and-paste a few characters from web pages.

This table is indexed by the Unicode number, in hexadecimal notation. The first 32 of the 128 (`U+0080`–`U+009F`) are control codes and `U+00A0` is a non-breaking space, so is invisible, while `U+00AD` is a soft hyphen (which we have not implemented and so is excluded).

Table 4.1.5 Latin-1 Supplement, Unicode `U+00A0`–`U+00FF`

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00A_		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
00B_	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
00C_	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
00D_	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
00E_	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
00F_	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

4.1.4.6 Arithmetic

If you are writing about technical subjects, then you will want to avail yourself of PreTeXt's extensive support for mathematics. Otherwise, you may wish to write *really simple* arithmetic within sentences without extra formatting. Notice that there is no provision for preventing line-breaks in the midst of these expressions.

So you can author (2×6)÷3+10−15 = -1, but that is about the limit of the complexity of expressions you should author without using the extensive capabilities designed for *mathematics*, rather than *arithmetic*. Note that the spaces around the equal sign have been supplied in the source, but no spaces have been provided around the operators. Also, the minus sign and the negative are slightly different because the subtraction uses the `<minus/>` element, while the negative answer uses a plain keyboard hyphen/dash.

Using the `<m>` element instead, the above is $(2 \times 6) \div 3 + 10 - 15 = -1$. Note the more careful spacing, and the appropriate symbols for subtraction and negation, with no special care in the LaTeX syntax used in the source.

Note also that the plus sign, + and the equals sign, =, can be provided in text as the unambiguous keyboard characters.

The `<degree/>`, `<prime/>`, `<dblprime/>` elements support simple coordinates with degrees, minutes, seconds, or temperature, or distance in feet and inches. "We parked the car at 36°16'0.83''N, 122°35'47.27''W, and since it was 93°F, we walked 505'3.6'' so we could swim in the bay."

`<minus/>`, −, minus, subtraction, negation. For simple arithmetic expressions in text, this symbol may be used. Note that the keyboard hyhpen (or dash) might be acceptable for your purposes, but they are different.

`<times/>`, ×, times, multiplication. For simple arithmetic expressions in text, this symbol may be used. Or it may be used to specify dimensions, as in "I bought a 2×4 at the lumber yard."

`<solidus/>`,⁄, solidus, virgule, fraction bar. For simple arithmetic expressions in text, this symbol may be used to form a fraction. It should appear to have a significantly shallower slope than the forward slash, /.

`<obelus/>`, ÷, obelus, division sign. For simple arithmetic expressions in text, this symbol may be used to indicate division.

`<plusminus/>`, ±, plus-minus sign. For simple arithmetic expressions in text, this symbol may be used to indicate a tolerance or a choice of two values, one the negative of the other.

`<degree/>`, °, degree symbol. A raised open circle for temperature or for angles used in coordinates.

`<prime/>`, ', prime symbol. A straight mark that is placed like an exponent. For use in coordinates or statements of linear measure in feet and inches. Not an apostrophe, and not mathematics (like, say, not to denote a derivative).

`<dblprime/>`, '', double prime symbol. Two straight marks that are placed like an exponent. For use in coordinates or statements of linear measure in feet and inches. Not an apostrophe, and not mathematics (like, say, not to denote a second derivative).

4.1.4.7 Separators

`<ndash/>`, –, en dash. A dash, the width of a lowercase 'n', or exactly half the width of the em dash. This is typically used to express a range, such as `1955<ndash />1975`, with no intervening spaces. It is often expressed as two hyphens when typed. Bringhurst suggests an ndash surrounded by spaces – thusly – when setting off phrases.

`<mdash/>`, —, em dash. A dash, the width of a lowercase 'm', or twice the width of the en dash. This is typically used to express a secondary part of a phrase, much like the use of a semi-colon or parentheses.

Style guides suggest that there should be no spaces, before or after, an em dash, while some allow for a "thin" space on either side. You should always leave no space around an `<mdash/>` element in your PreTeXt source. Then a publication file entry can be used to elect the automatic addition of a thin space, should your publisher so desire. See Subsection **??** for the syntax of the publisher file entry.

`<nbsp/>`, non-breaking space. A space, but which ties two words together and discourages a line break when formatted, such as `Summer<nbsp />1967`. This can also be used to discourage a period in an abbreviation from being interpreted as the end of a sentence, such as `C.R.<nbsp />Darwin`.

`<midpoint/>`, `<swungdash/>`, ·, ~, midpoint, swung dash. These can be used·as more decorative ~ separators.

4.1.5 Keyboard Keys

If you are writing a software manual, or writing about how to use a calculator as part of a science textbook, then you might want to make it very clear which keys to press on a keyboard. The `<kbd>` element can hold content like `Z` or `Caps Lock` and your output will have a very nice looking keyboard key with the desired label. For example, Z and Caps Lock. For keys labeled with graphics, like the arrow keys, instead of content, provide the `@name` attribute with a value from the following table. Request additions to this table if you are using this feature (2019-11-22).

Table 4.1.6 Keyboard Keys Specified by `@name` Attribute

`left`	←	`right`	→	`up`	↑	`down`	↓
`enter`	⏎	`shift`	⇧	`solidus`	/		
`plus`	+	`minus`	−	`times`	×	`left-paren`	(
`obelus`	÷	`squared`	x^2	`inverse`	x^-1	`right-paren`	)

4.1.6 User-Interface Icons

When writing about software, you may wish to call attention to icons used in the interface. Similar to graphical keyboard keys, use the `<icon>` element with a `@name` attribute with a value from the following table. See Section **??** for information about using icons as part of LaTeX output.

Table 4.1.7 User-Interface Icons available with `<icon/>`

Name	Icon	Name	Icon	Name	Icon
`file-save`	💾	`arrow-left`	⬅	`media-rewind`	⏪
`arrow-up`	⬆	`arrow-right`	➡	`media-fast-forward`	⏩
`menu`	☰	`arrow-down`	⬇	`media-skip-to-start`	⏮
`wrench`	🔧	`media-stop`	■	`media-pause`	⏸
`gear`	⚙	`media-play`	▶	`media-skip-to-end`	⏭
`power`	⏻				

4.1.7 Other Markup in Paragraphs

`<today/>`, `<timeofday/>`, March 11, 2024, 10:02:04 (-07:00). Values at the time of XML processing. Useful for marking drafts or other frequently revised material such as online versions.

`<tex/>`, `<latex/>`, `<xetex/>`, `<xelatex/>`, `<pretext/>`, `<webwork/>`, TeX, LaTeX, XeTeX, XeLaTeX, PreTeXt, WeBWorK. Conveniences for frequently-mentioned accessories to PreTeXt.

`<fillin/>`, _____, fill-in blank. A “fill in the blank” blank. May be used in normal text or within mathematics contexts. The `@characters` attribute may be used to hint at how long the line will be. Here we have set `@characters` to the value 5.

If used inside math, a `@fill` attribute (in lieu of `@characters`) may be some string of math that will be used to determine width, height, and depth of the blank. In text, the default value of `10` is used for `@characters`. In math, the default value of `XXX` is used for `@fill`.

If used in normal text, a `@rows` and/or `@cols` attribute may be present, each a positive integer. When either is greater than one, there will be an indication that the expected content to fill in the blank is a `@rows` × `@cols` array.

`<ie/>`, `<eg/>`, `<ca/>`, `<vs/>`, `<etc/>`, i.e., e.g., ca., vs., etc.. A small collection of frequently-used Latin abbreviations, with attempts to handle the tricky periods wisely in LaTeX output. Strictly speaking BC is not Latin, but we include it for completeness. Tags are always lowercase, no punctuation, usually two letters.

Tag	Realization	Meaning
`ad`	AD	*anno Domini*, in the year of the Lord
`am`	AM	*ante meridiem*, before midday
`bc`	BC	English, before Christ
`ca`	ca.	*circa*, about
`eg`	e.g.	*exempli gratia*, for example
`etal`	et al.	*et alia*, and others
`etc`	etc.	*et caetera*, and the rest
`ie`	i.e.	*id est*, in other words
`nb`	NB	*nota bene*, note well
`pm`	PM	*post meridiem*, after midday
`ps`	PS	*post scriptum*, after what has been written
`vs`	vs.	*versus*, against
`viz`	viz.	*videlicet*, namely

SI Units. *Système international (d'unités)* (International System of Units) is a system of measurement used universally in science. PreTeXt has comprehensive support for this system and its notation and abbreviations. See Section 3.27 for a short introduction and Section ?? for detailed descriptions of the relevant elements and their use.

𝄪, ♯, ♮, ♭, 𝄫, music notation. Notes, chords, and other notation may appear within sentences as part of a discussion. See Section ?? for detailed descriptions of the relevant elements.

Best Practice 4.1.8 Understand the Importance of Careful Markup. There is a lot of detailed information in this section. Much of it is critically important. If you are new to thinking in terms of markup (rather than WYSIWYG tools), it might be overwhelming, a lot to digest, and hard to separate the wheat from the chaff. Careful here means using the necessary markup, not using it for other purposes different than its intent (**tag abuse**), planning ahead for different output formats, but not becoming a slave to over-doing it.

So come back here often for a re-read. And keep in mind that PreTeXt is designed around principles (List 1.1.1), and that it is markup (Item 1.1.1:1) which enables multiple outputs (Item 1.1.1:3) and effective and beautiful online versions (Item 1.1.1:6).

4.2 Blocks

4.2.1 Introduction to Blocks

A division (Section 3.1), that is not further subdivided, is primarly, but not exclusively, composed of paragraphs and **blocks**. We document the types of blocks here, even though we do not intend to ever provide a rigorous definition of the term. Here is a list of characteristics, which is not prescriptive.

List 4.2.1 Characteristics of a Block

- Visually set-off from a run of "plain" paragraphs. Often earning a number, and ideally provided a title or caption (Best Practice 4.8.1).
- Reflowable lines of text, such as an `<example>`, *or* a more rigid, more spatial, more **planar** object, such as a `<figure>` or `<table>`.
- Usually a child of a division (Section 4.6). But see just below.
- Typically a block does not contain another block, except that the more planar ones can appear as part of a more textual one.

- When numbered, all blocks generally run consecutively. (In LaTeX-speak, "on the same counter.") Numbering FIGURE-LIKE and PROJECT-LIKE independently is in transition at this writing (2021-07-07).
- Depending on the output format of your document, some block types may be initially hidden to improve the visual flow. The reader must click on the heading for the block to reveal its contents. You can change this behavior by configuring your publication file. See Section **??**. We will make note of the elements that are hidden by default for HTML output.

The following is somewhat general, and we have not extensively cross-referenced to the particular types of blocks, so use the Table of Contents or the Index to learn more specifics.

4.2.2 Isomorphic Blocks

The structure of a block is described carefully in the schema (Chapter **??**). There are approximately forty blocks that are arranged into ten groups, within which they behave identically, except for their displayed names. An exception is the group of four "figure-like" items which are very similar, but have differences beyond just their displayed names. These groupings are defined in the `xsl/entities.ent` file, which we summarize in the next table. The category name is taken from the entities file, and the notes are meant to describe the distinctive capabilities of the category.

List 4.2.2 Summary of Blocks

REMARK-LIKE	`<remark>`, `<convention>`, `<note>`, `<observation>`, `<warning>`, `<insight>` The most basic, generic, block.
EXAMPLE-LIKE	`<example>`, `<question>`, `<problem>` A worked problem meant as exposition. It can be structured with `<task>`, `<hint>`, `<answer>`, and `<solution>` just like an `<exercise>` or PROJECT-LIKE, but the `<hint>`, `<answer>`, and `<solution>` cannot be electively removed from output, and they do not migrate to collections of solutions elsewhere. Hidden by default for HTML output formats.
PROJECT-LIKE	`<project>`, `<activity>`, `<exploration>`, `<investigation>` These are similar to an `<exercise>`, but the name suggests a slightly different undertaking, and they cannot be placed in an `<exercises>` division. The current default is that they are numbered independently from other blocks, but this is planned to switch to elective behavior. The `<hint>`, `<answer>`, and `<solution>` behave more like those for an `<exercise>` and can be removed from output, and can migrate to collections of solutions.
FIGURE-LIKE	`<figure>`, `<table>`, `<listing>`, `<list>` An object that is a container for other atomic objects, which are typically somewhat rigid (not reflowable) or two-dimensional. Typically with a number, and provided with a title or caption. But each is slightly different in what it can contain and how it is rendered. A `<table>` can only hold a `<tabular>`, while a `<listing>` is meant for `<program>` and `<console>`. A `<list>` is not the list itself, but a container for one of the three possible lists (see Section 4.11 and especially Section 4.20). A `<figure>` is the most liberal, allowing a wide variety of contents, with `<image>` being the prototypical example. These are not designed with the expectation that they can be renamed.
THEOREM-LIKE	`<theorem>`, `<corollary>`, `<lemma>`, `<algorithm>`, `<proposition>`, `<claim>`, `<fact>`, `<identity>` Mathematical results, which can have an (optional) `<proof>`. Proofs are hidden by default in HTML output.

AXIOM-LIKE	`<axiom>`, `<conjecture>`, `<principle>`, `<heuristic>`, `<hypothesis>`, `<assumption>` A mathematical statement, which does not have a proof.
DEFINITION-LIKE	`<definition>` A definition of a (mathematical) object.
ASIDE-LIKE	`<aside>`, `<biographical>`, `<historical>` Parenthetical content that is structured beyond what a footnote can contain.
COMPUTATION-LIKE	`<computation>`, `<technology>`, `<data>` For descriptions of activities or data for use with computers, calculators, or other devices.
GOAL-LIKE	`<objectives>`, `<outcomes>` These are structured primarily as lists, and may only appear early (`<objectives>`) or late (`<outcomes>`) within a division.

4.2.3 Other Blocks

There are other blocks which can be achieved using just one element. Examples are `<poem>` and `<assemblage>`. An `<exercise>` can take on different behaviors, depending on its placement (see Section 4.3). One such placement is as a child of a division, which we call an **inline exercise**, and this would be regarded as a block, very similar to a PROJECT-LIKE. Inline exercises are hidden by default in HTML output.

A paragraph (`<p>`) can appear many places and is a primary component of blocks. But when it is a child of a division, it shares some charateristics with other blocks. There are a number of peers of a `<p>` which would then also qualify: `<pre>`, `<blockquote>`, `<image>`, `<video>`, `<program>`, `<console>`, and `<tabular>`. None of these can have a title or number, making them less useful, but widening the possibilities for placement.

4.2.4 Renaming Blocks

A common desire of authors new to PreTeXt is a new block. Authors extending PreTeXt to add a new one is not supported, and it is not even straightforward for a PreTeXt developer to provide comprehensive support for a new block. One of the reasons for multiple versions of blocks, with common behaviors, is that you can appropriate one for use with your project and give it a new name. So for example, a biology textbook might want to use `<activity>` for a laboratory, but rename it to “Laboratory”. Of course, this means you will forego having any “Activity” in your book. (But if you *are the first* to write a biology textbook in PreTeXt perhaps you should talk to us about a real `<laboratory>` element that behaves well for all the physical and biological sciences!) Here is the procedure:

1. Choose a block from List 4.2.2 that has behavior similar to your intended use, and which you do not forsee using as-is in your project.
2. In the `<docinfo>` section use a `<rename>` element. The content should be the new name (“Laboratory” above), while the `@element` attribute should be the name of the element renamed (`activity` above).
3. The `@xml:lang` attribute should be used to specify a value of the code for use with documents authored in languages other than US English. If absent, the default vslue `en-US` is used. Multiple `<rename>` elements for the same element, in different languages, is supported.

So, continuing and extending our example, an author would use

```
<rename @element="activity" xml:lang="fr-FR">Laboratoire</rename>
```

The most popular block to rename is `<exercise>`. We have exercises inside `<exercises>` divisions, which we call **divisional exercises**, and exercises inside divisions, which we call **inline exercises**. It is possible to have one of each that have identical numbers. So in cross-references the former is an “Exercise”, while the latter is a

"Checkpoint". If you only have inline exercises, you might prefer that they be called Exercise rather than Checkpoint. For the `@element` attribute of the `<rename>` element, use a **pseudo-element**, in this case `inlineexercise`. Other pseudo-elements which can be used to distinguish the various types of exercises are: `divisionexercise`, `worksheetexercise`, and `readingquestion`.

Choose the element you rename carefully, trying to match it to the purpose of your content. It can be useful for other purposes, such as automatic lists (Section 4.28), and you may decide later to use other properties of the element you have chosen.

4.3 (*) Exercises, Inline and Divisional

4.4 Verbatim and Literal Text

This section expands on parts of Section 3.16. For descriptions of more involved uses, such as program listings and console sessions, see Section 4.15.

The tags described here contain *only raw characters*. By that we typically mean the first 128 characters of the ASCII code. Unicode characters are likely to migrate to HTML output just fine, but results for LaTeX output will be variable. The restriction to character data has two consequences. First, any markup mistakenly included will have its content silently ignored and dropped. Second, you need to observe the rules on exceptional characters and escaped characters for XML for literal text, which are mercifully simple.

> In your source, use `&amp;` for an &, use `&lt;` for <, and optionally use `&gt;` for >. Otherwise, every other ASCII character will render faithfully across all possible formats.

See Subsubsection 4.1.4.2 for more detail and explanation.

4.4.1 Short, Inline, Verbatim Text

The `<c>` tag is a mnemonic for "code", but is really meant to be any chunk of literal characters that you want to emphasise that way. So a "typewriter" font of fixed-width characters would be a typical presentation. It is meant for use within a sentence or caption ("inline") so its use is limited to those situations, and others that are similar, such as a title or a cell of a table. Typically these pieces of text do not hyphenate words, and so can lead to spillover into a right margin. In these situations, consider options below for longer pieces of text.

4.4.2 Longer, Inline, Verbatim Text

For longer pieces of verbatim text, use the `<cd>` tag, which is short for "code display", analogous to the `<md>` for mathematics. It is used within sentences of a paragraph and will be presented with a vertical break above and below, but without interrupting the paragraph. Because of the display presentation, it cannot be used other places, such as a `<title>`, where a vertical gap is not appropriate. All of the previous discussion about exceptional characters applies for this tag.

You have two options in use. You may author inline with the rest of a sentence, with no extra newlines or whitespace before, after, or within the content. The result will be a single displayed line.

Or you may structure the content using one, or more, of the `<cline>` tag, which is meant to be similar to the `<line>` tag used elsewhere. You should still take care to not place any extra whitespace before or after the `<cd>` element, but in between the `<cline>` you may use as much visual formatting of your source as you wish, especially if you like your source to mirror your output.

4.4.3 Blocks of Verbatim Text

If you want to isolate large chunks of verbatim text outside of paragraphs, the `<pre>` tag is the one to use. It can be used as a peer of paragraphs (and other structures) as a child of a division, or it can be placed into a `<listing>` to receive a caption, title and number.

You can structure the contents with `<cline>` in exactly the same manner as for `<cd>`. But you may find this tedious. Instead, you can make the content of `<pre>` a sequence of lines separated by newlines. So that you can preserve the indentation of your source, the line closest to the left margin is taken to actually be the left margin, and a corresponding amount of leading whitespace will be removed from every line. This will work well if you recognize two caveats. First, results will be unpredictable if you mix spaces and tabs for indentation. Sticking with spaces is best. Second, if your first characters of content immediately follow the `<pre>` tag then there is no leading whitespace and it is as if that line is already at the left margin. Then subsequent indentation may seem too severe to you.

As previously mentioned, Section 4.15 discusses the `<console>` and `<program>` tags which are more specific, and hence more capable. Review the possibilities before you decide between `<pre>`, `<console>`, and `<program>`.

4.5 Cross-References and Citations

When you read a novel, you would likely read it cover-to-cover (in one sitting?) and then put it away and never read it again. But for a textbook, you may read cover-to-cover, but you may also frequently skip around, especially at exam time. And once read, it might become a reference work for you, since you know it so well. So years later you might come back looking for something. Cross-references help with all this, so use them liberally. Recognize that an index is really just a specialized way to provide an abundance of cross-references.

4.5.1 Creating a Cross-Reference

It is a two-step process to make a cross-reference.

1. Put an `@xml:id` attribute on any element you think you might want to reference later. Be organized about the values of these attributes, and in particular do not number them, as this has no place in your source, and you do not want to maintain the changing numbers over the life of your project. See the advice in Section 3.4 about banned characters. Some elements do not accept this attribute because the element has nothing to identify it (no number, no title). Typically these are **containers** such as `<sidebyside>`, `<statement>`, or `<ol>`. In these cases, put the attribute on the closest enclosing element.

2. To make a cross reference, you create an `<xref/>` element with a `@ref` attribute with the same value as `@xml:id` attribute on the element you want to reference.

Simple. It is meant to be, so you can use it liberally. But we also know authors want some flexibility.

Best Practice 4.5.1 Use `@xml:id` Frequently. Use the `@xml:id` attribute liberally, on any object you might want to reference later, and on every division. It is easier to do as you author and will be very valuable later. (Trust us on this one.) Develop a system so you can recall them predictably, but keep them readable. Don't use numbers, they will change. Then make frequent cross-references. They are relatively easy for you and will be incredibly useful for each and every one of your many readers, over and over and over again.

4.5.2 Text of a Cross-Reference

By default, a cross-reference will visually be text like Theorem 5.2. Depending on your output format, it may have various devices to help you locate that theorem. Maybe a page number, or it is a hyperlink, or the whole theorem is the content of a knowl. You can change the default look of cross-references by setting the `@text` attribute in `docinfo/cross-references`. But you can also change the visual appearance of a cross-reference on a case-by-case basis. Add a `@text` attribute to your `<xref/>` element to override the document-wide setting. The first column of this table lists the possible attribute values, either document-wide, or on a per-cross-reference basis. The second column has live cross-references to a section of an earlier chapter (that is far away). The third column has live cross-references to another section of this chapter (which is close by). For the fourth column, we have placed content ("Extra") into the `<xref>` element as an override of, or addition to, some of the text for the cross-references of the preceding column. Study the table and then read some more explanation following. Note that `type-global` is the default.

Table 4.5.2 Cross-reference visual text styles

`@text`	Far Away	Close By	With Content
`type-local`	Section 5	Section 6	Extra 6
`type-global`	Section 3.5	Section 4.6	Extra 4.6
`type-hybrid`	Section 3.5	Section 6	Extra 6
`local`	5	6	Extra 6
`global`	3.5	4.6	Extra 4.6
`hybrid`	3.5	6	Extra 6
`phrase-global`	Section 5 of Chapter 3	Section 6 of Chapter 4	Warning
`phrase-hybrid`	Section 5 of Chapter 3	Section 6	Warning
`title`	Titles	Divisions	Warning
`custom`			Extra

Note that `local/global` describes the uniqueness of the number (and is affected by your choice of numbering schemes), while `type` refers to an automatic prefix of the number. The text of the type will vary according to the document's language. If a cross-reference and its target are close to each other, a number like 5.8.2.4 might be overkill, when just a 4 would suffice. A `hybrid` scheme will use the shorter number whenever it makes sense. There are two `phrase` schemes, and it should be clear what text `title` will produce (though realize there must be a title for the object, possibly a default provided by PreTeXt). Finally, if desired, the text can be customized with any text you like.

You can also override the text used in a cross-reference link. You do this by providing content to the `<xref>` element. In other words, do not use an empty tag, but put some (simple) text in the element. Generally, this additional text becomes a prefix of a number or a replacement of a type. It is better to use these overrides, since in electronic formats, the text of the override will be incorporated into the "clickable" portion of the resulting link, making a larger item to hit. Recognize that this extra content will not benefit from automatic internationalization.

Here are careful examples of providing content inside the `<xref>` element, where we have provided the content "Division" in the live examples. The list is not exhaustive.

Table 4.5.3 Cross-reference text overrides

`@text`	Example
`'global'`	Division 4.5
`'type-global'`	Division 4.5
`'custom'`	Division

4.5.3 Variations

There are some variant uses for the `<xref>` tag.

- Replace `@ref` by `@provisional` and give it a value with some simple text like `subsection on eagle habitat` and you will get reminders that once you write this future subsection you need to link it in right here. This is just a convenience for authors during the early stages of a writing project (see Section **??** for details).

- Replace `@ref` by the pair `@first` and `@last`. Provide attribute values just as you would for `@ref`. The code will check that the targets have the exact same tag, and that the order is correct. You will get a link that looks like a range, separated by an en dash. As a link, only `@first` will be used for the linking mechanism (i.e., one link is generated, not two). Experiment with `@text` and content overrides.

- The `@ref` attribute may be a list of `@xml:id`, separated by commas or by spaces. Then you will get back a list of cross-references. This is meant for a list of citations, producing a look like [5, 9, 22], but it makes no restriction to this case. Use it generally, but it is unlikely to get any more capable. If you want a different list, just use multiple `<xref>` and format as you desire.

You can create many different combinations with the text options and the variants. Here is one example. You want to say Chapters 1–**??**. As a range you use the variant with two references. You would get "Chapter" out front automatically with the `type-global` scheme, but a plural form makes more sense. So you use that text as an override.

We could use either `type-global` or `global` to get the same text, and possibly `type-hybrid` depending on the place where you built the cross-reference. So possibly, one of these schemes might be your document-wide setting and you do not need to specify it now. Here is what we just used:

```
<xref first="start-here" last="schema" text="global">Chapters</xref>
```

You can place a cross-reference into a `<title>` element, but a particular conversion is free to simply render it as text, and not as an active link.

Finally, there is fairly robust error-checking to protect against typographical errors in the attribute values that need to match up for all this to work. Also, there is a check that the `@xml:id` are unique. But all this checking happens at processing-time, not at the validation step. Any suggestions for improvements that make these checks even more robust are welcome.

4.5.4 Citations

Citations are just specialized cross-references to `<biblio>` elements, and so behave the same way as other cross-references. However they will always visually look like [23], and there is no notion of "type name."

4.5.5 Equations

Similar to citations, references to equations (`<men>` and `<mrow>` elements) will visually look like (4.2), where the type name is implied by the parentheses. Notice that you cannot cross-reference an `<me>` element (it has no number) or an `<md>` element (it is just a container, filled with `<mrow>` that you can target if you give them numbers). Consult Subsection 4.9.3 for details about controlling the numbering of equations within an `<md>` or `<mdn>` element.

4.5.6 Lists

Roughly, you can target a list item for a cross-reference, but not the list itself, since it is a container. See Subsection 4.11.4 for precise details about using list content as the target of a cross-reference. Note also, that an entire named list may be the target of a cross-reference, see Section 4.20. Here we concentrate on the text of the cross-reference itself.

First, note that if you cross-reference a list item of an anonymous list, there is a very real possibility that the number will be ambiguous, and there is no option for `@text` will save you, and never can be. See the middle column of Table 4.5.5 for the demonstration. We assiduously try to make it *impossible* to ever create ambiguous text for cross-references, especially in consideration of print output. Use the feature carefully.

Best Practice 4.5.4 Take Care Referencing Anonymous Lists. Cross-references to list items of anonymous lists can easily be ambiguous and then useless for readers of print. Keep such a cross-reference close to its target, ideally within the same list, and/or perhaps using additional, unambiguous clues about location (which you expect will survive later editing):

1. See Item 2 of this list.
2. See Item 1 in the list appearing in Best Practice 4.5.4.

The `local` option, discussed generally above, behaves differently for a cross-reference to a list item of an ordered list that is contained in a named list. As seen in the table just below, the local portion of the number is the part that comes from the list item, without the part that comes from the location of the `<list>` block.

Table 4.5.5 Cross-references to list items, visual text styles

`@text`	Named List	Anonymous List	With Content
`type-local`	Item B.c	Item 2.III	Extra B.c
`type-global`	Item 4.11.4:B.c	Item 2.III	Extra 4.11.4:B.c
`type-hybrid`	Item 4.11.4:B.c	Item 2.III	Extra 4.11.4:B.c
`local`	B.c	2.III	Extra B.c
`global`	4.11.4:B.c	2.III	Extra 4.11.4:B.c
`hybrid`	4.11.4:B.c	2.III	Extra 4.11.4:B.c
`phrase-global`	Item B.c of List 4.11.4	Warning	Warning
`phrase-hybrid`	Item B.c of List 4.11.4	Warning	Warning
`title`	A Test Title	Exquisite Fish	Warning
`custom`			Extra

The `hybrid` options employ a different definition of when the distance between a cross-reference and its target is close enough that the number can be shortened, without becoming ambiguous. When an `<xref>` and its target are part of the same `<list>`, then the common part of the number derived from the `<list>` is not needed. To illustrate we need to make a small `<list>` with cross-references contained within.

List 4.5.6 Small test

1. (`type-global`) Flowers are not Item 4.5.6:6.
2. (`type-hybrid`) Fish are not Item 6.
3. (`hybrid`) Bacteria are not 6.
4. (`phrase-global`) Fungi are not Item 6.
5. (`phrase-hybrid`) Trees are not Item 6.
6. Mammals.

Best Practice 4.5.7 Be Rational About Numbering Variations. With distinct numbering schemes for divisions, theorems, figures, equations, and citations, along with ten different text styles for a cross-reference, plus variants, per-cross-reference settings, and text overrides, there is a huge supply of combinations. Likely you can create some really ugly cross-references. Use the flexibility sensibly.

4.6 Divisions

A **division** (or more carefully, a **structural division**) is a structured component of a book or article that would be recognized by most any reader. They are essential to the organization of a PreTeXt project. Notice that we use the generic term division, since a `<section>` is just one example of a division.

Divisions are `<book>`, `<article>`, `<part>`, `<chapter>`, `<section>`, `<subsection>`, `<subsubsection>`, and `<paragraphs>`. Their use is fairly intuitive, though there are some restrictions, so please read on.

A `<book>` must contain at least one `<part>` or at least one `<chapter>`, which may contain `<section>`, `<subsection>`, and `<subsubsection>`. A `<part>` simply contains a sequence of `<chapter>` and functions in two user-selectable ways: structural (e.g. numbering will reset), or decorative (merely inserting a decorative page between two chapters and sectioning the Table of Contents).

An `<article>` is simpler and shorter than a book. It might be really simple and have no divisions at all, or it may have `<section>`s. It cannot have `<chapter>`s, as that would be a `<book>`. Within a `<section>`, `<subsection>`s and `<subsubsection>`s may follow.

Divisions must nest properly and may not be skipped. So a `<section>` cannot contain a `<chapter>` and a `<subsection>` may not be contained in a `<chapter>` without an intervening `<section>`.

A division *must* contain a `<title>`, and may contain one or more index entries (see Section 4.26), which should appear before anything else. Any division may be unstructured, with just a sequence of **top-level content** such as paragraphs, figures, lists, theorems, etc. Or a division may be structured, and in this case it must follow a prescribed pattern. There may be a single, optional `<introduction>`, filled with top-level content, followed by a sequence of at least one of the appropriate divisions, ending with a single, optional `<conclusion>`, filled with top-level content. It is an error to begin with a run of top-level content inside a division and then begin to use divisions. (The solution is to make the initial content an `<introduction>` and/or one or several divisions.)

A `<book>` may be structured into parts. After the `<frontmatter>` and before the `<backmatter>` there may be a sequence of `<part>`. These elements may carry a `<title>`, and not much else, besides a substantial sequence of `<chapter>`. The main effect is to get an extra level of division in the Table of Contents. For print and PDF there is an entire page devoted to the title and number of the part. The default numbering is **decorative**, which means that the chapters are numbered consecutively from the start of the book and do not reset to one at the start of each part. It is as if the parts were not even there. The alternative is that parts are **structural**. Now each part begins with Chapter 1. There are other more subtle differences, such as cross-references use a part number if, and only if, the trip from the cross-reference to its target crosses a boundary of two parts. The two approaches to part structure may be set via the publication file, see Section **??**.

There are exceptions to the above. For one, `<paragraphs>` is an anomalous division, as a sort of lightweight sectioning command. It may appear in any division, at any location within a division, it may not be divided further (it is a leaf of the document tree), it never gets a number, and its title is formatted in a subsidiary way. I especially like to use this in a two- or three-page `<article>` that has no other divisions at all. It is also very useful as a way to subdivide portions of the front matter (Section 4.24), such as a `<preface>`. Typical presentation has the title in bold, without much change in font size (if at all), inline with the first paragraph, and perhaps a bit of vertical space as it begins and ends. Despite the name, it may contain more than just paragraphs, so may contain any top-level-content that would go in any other division.

4.7 Specialized Divisions

There are six divisions that have specialized functions, and therefore have less generic names than ones like `<chapter>` or `<section>`. They are `<exercises>`, `<reading-questions>`, `<solutions>`, `<references>`, `<glossary>`, and `<worksheet>`. They have some features in common, such as having a `<title>`, but each is different from the other in substantial ways.

Generally, a specialized division may be placed within any other division (Section 4.6), and it will behave like a subdivision of that division. Some may be placed in the back matter and may behave as a version relevant to the entire document. This section describes the specifics for each type of specialized division.

4.7.1 (*) References (Lists of Works Cited)

4.7.2 Glossary

A glossary is a list, in alphabetical order, of foreign or unfamiliar words, along with definitions. [1, 1.87]

A `<glossary>` division may be placed inside any main matter division, and at most once in the `<backmatter>`. In either event, the *Chicago Manual of Style* [1, 1.87] indicates that it should be placed right before a `<references>`.

After a `<title>`, index entries, and other metadata, a `<glossary>` division may begin with an optional `<headnote>`, which can use paragraphs to explain anything unusual about the construction of a particular glossary.

The remainder of a glossary is a sequence of items to explain. Typically these are words, phrases, initialisms, or acronyms. Each item is a “glossary item”, enclosed in a shorthand `<gi>` element. The element must lead with a `<title>`, which is the term being explained. PreTeXt will provide a period after each defined word, so there should not be any punctuation in your source at this location. The term should not have any markup, *unless* the markup is used in every occurence of the term in the text. Similarly, a term is capitalized only if it is capitalized routinely in the text.

The explanation itself follows, typically in a sequence of paragraphs, but unnumbered items, such as an `<image>` may also be used. It is the author’s responsibility to create the list in alphabetical order. Automatic groupss (according to initial letter) are a pending feature request, perhaps especially for a final, overall, back matter glossary, much like an index. See GitHub #1971[1].

[1] `github.com/PreTeXtBook/pretext/issues/1971`

Many of the preceding recommendations can be found in *Chicago Manual of Style* [1, 2.28]. For an example, see the glossary in the back matter of this Guide.

4.7.3 (*) Worksheets

4.8 Titles

Divisions always need titles, you accomplish this with a `<title>` tag first thing. Almost everything that you can use in a paragraph can be used in a title, but a few constructions are banned, such as a displayed mathematical equation (for good reason). Try to avoid using footnotes in titles, even if we have tried to make them possible.

A division will also support a single optional `<shorttitle>` and/or a single optional `<plaintitle>`. The full `<title>` will be used where the division is born, but in other places where the title is used for navigation, such as a Table of Contents, print page header, or HTML summary page, when horizontal space may be at a premium, the `<shortitle>` will be used preferentially.

A `<plaintitle>` is similar, but slightly different, so you might want both. In limited situations, PreTeXt markup does not travel well in certain conversions. The best example is mathematics, which might be in a title of a division, and then in a conversion to HTML, will fail to render in the tab of a browser, or the list of open tabs for a browser window. Two examples

```
<title><m>q</m>-crystalline cohomology</title>
<title><m>\delta</m>-rings</title>
```

PreTeXt will automatically do a nice job for you with the first, but the second will retain `\delta` in a browser tab. However, you can add a `<plaintitle>` element where the LaTeX `\delta` can be replaced by a Unicode delta (`U+03B4`), which will used preferentially and be fine in HTML output.

To avoid confusing your readers, use these alternate titles sparingly, and ideally only when you have a really, really, really long title and then use a short title that is easily recognizable as a variant of the long title, or when markup is behaving poorly in situations such as a browser tab. Your first option should be to ask if your long title must absolutely be so long, or if the markup is strictly necessary.

Many, many other structures admit titles. Experiment, or look at specific descriptions of the structure you are interested in. Titles are very integral to PreTeXt, much like cross-references. Titles migrate to the Table of Contents, get used in page headers for print output, can be used in lists (such as a List of Figures), and can be used as the text of a cross-reference, instead of a number. You might not be inclined to give a `<remark>` a title, but it would be good practice to do so. If you use knowls in your HTML output for structures such as `<example>` (or if somebody else may someday choose to), then your readers will be spared a lot of confusion if you supply informative titles for each. Your electronic outputs will be much more useful to your readers if you routinely title every structure that allows it (perhaps excepting `<exercise>` which can be known by their number).

If a title is very long, the `<line>` element can be used to indicate how the title should appear on multiple lines. Note: does not apply to all output formats.

Best Practice 4.8.1 Provide Informative Titles Liberally. Provisions for titles in many situations is a key PreTeXt feature. And then they are used for various purposes to benefit readers. A good example is when the HTML conversion is used to place content in a **knowl**, a unit of content that begins hidden, but can be revealed (and hidden again) with one click of a mouse. Since a reader cannot see the content originally, we will migrate a title into the clickable text. But we cannot read your mind—it is your job as the author to provide a title, and to provide a good one.

Even if you are not yet sure what a knowl is, and even if you think you do not want to use them, there are other good reasons to have a title (such as automatic lists, see Section 4.28). Constructing them on-the-fly is much easier than making a big chore out of going back and doing it later.

Bad	Example 5.10
Better	Example 5.10 A cool lizard trick.
Best	Example 5.10 Various colors and markings of a chameleon.

4.9 Mathematics

As mentioned in the overview, Section 3.6, we use LaTeX syntax for mathematics. In order to allow for quality display in HTML, and other electronic formats, this limits us to the subset of LaTeX supported by the very capable MathJax[1] Javascript library. Generally this looks like the `amsmath` package maintained by the American Mathematical Society at their AMS-LaTeX page[2]. For a complete and precise list of what MathJax supports, see MathJax's Supported TeX/LaTeX commands[3]. Once you have digested this more general section, be sure to also consult Section 4.10 for some very specific suggestions.

4.9.1 Inline Mathematics

Use the `<m>` to place variables or very short expressions within a sentence of a paragraph, the content of a `<title>`, a `<cell>` of a table, a footnote, or other similar locations of sentence-like text. You can't cross-reference this text, nor make a knowl with it. Though you can typically cross-reference a containing element.

Do not use LaTeX-isms like `\displaystyle` to try to end-run the inline nature. It will just lead to poor results.

Best Practice 4.9.1 Keep Inline Mathematics Short. Longer mathematical expressions in an `<m>` element can lead to awkward line breaks, both in HTML output, and especially in PDF generated from LaTeX. And complicated fractions or integrals can introduce abnormal line-spacing that is distracting to a reader. As a rough rule-of-thumb, keep an inline expression shorter than a moderately-long regular word and avoid tall constructions. This should allow LaTeX's line-breaking algorithms the best chance of success.

So a simple, short equality such as $x = 2$ should not cause a problem, but if you want to claim that the probability distribution of the normal distribution has the right scaling factors, $\int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma}}\,dx = 1$, there is a good chance it will do less harm to your paragraph of you display it

$$\int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma}}\,dx = 1$$

using the `<me>` element described next.

4.9.2 One-Line Display Mathematics

The `<me>` element can be used for longer expressions or a single equation. Typically you will get vertical separation above and below, and the contents will be centered. See below about concluding periods (and other punctuation), and alignment. The `<men>` variant will produce a numbered equation, and therefore with a provided `@xml:id` attribute, can be the target of a cross-reference (`<xref>`).

4.9.3 Multi-line Display Mathematics

We begin with a pure container, either `<md>` or `<mdn>`. The former numbers no lines, the latter numbers every line. Within the container, content, on a per-line basis, goes into a `<mrow>` element. You can think of `<mrow>` as being very similar to `<me>`. In a LaTeX `align` the `\\` mark the end of a displayed line. In PreTeXt each `<mrow>` delimits a displayed line, and there are no `\\`s. Use `\amp` to mark the alignment point.

On any given `<mrow>` you can place the `@number` attribute, with allowable values of `yes` and `no`. These will typically be used to override the behavior inherited by the container, but there is no harm if they are redundant. A given line of the display may be the target of a cross-reference, though the numbering flexibility means you can try (and fail) to target an unnumbered equation.

An `<mrow>` may have a `@tag` attribute in place of a `@number` attribute. This will create a "number" on the equation which is just a symbol. This is meant for situations where you do not want to use numbers, and the resulting cross-reference is "local." In other words, the `<xref>` and its target are not far apart, such as maybe within the same `<example>` or the same `<proof>`. Allowable values for the attribute are:

[1]`www.mathjax.org`
[2]`www.ams.org/publications/authors/tex/amslatex`
[3]`docs.mathjax.org/en/latest/input/tex/macros/`

```
star, dstar, tstar,
dagger, ddagger, tdagger,
daggerdbl, ddaggerdbl, tdaggerdbl,
hash, dhash, thash,
maltese, dmaltese, tmaltese
```

These are the names of symbols, with prefixes where the prefix `d` means "double", and the prefix `t` means "triple". Cross-references to these tagged equations happens in the usual way and should behave as expected. See Section 3.4 and Section 4.5 for more on cross-references.

4.9.4 Exceptional Characters

The LaTeX macros, `\amp`, `\lt`, and `\gt` are always available within these mathematics elements, so that you can avoid the exceptional XML characters &, < and >. See Section 3.14 for this same information, but in the broader context of your entire document.

4.9.5 Text in Mathematics

Once in a while, you need a little bit of "regular" text within an expression and you do not want it to look like a product of a bunch of one-letter variables. Use the `\text{}` macro for these. Only. Other ways of switching out of math-mode and into some sort of "regular" text will appear inferior, and can raise errors in certain conversions.

- Do place surrounding spaces inside the `\text{}` macro.
- Do not place any mathematics inside the `\text{}` macro.
- Do not use the `\mbox{}` macro as a substitute.
- Do not use font-changing commands (e.g. `\rm`) as a substitute.

For example,

```
<me>f(x) = \begin{cases} x^2 \amp \text{if } x\gt 0\\
           -7 \amp \text{otherwise} \end{cases}</me>
```

produces

$$f(x) = \begin{cases} x^2 & \text{if } x > 0 \\ -7 & \text{otherwise} \end{cases}.$$

This example amply illustrates the use of macros for XML exceptional characters (twice), appropriate use of the `\text{}` macro (twice), spaces in the `\text{}` macro (once), sentence-ending punctuation (see the source, the period is *not* inside the `<me>` element) and yes, we did think twice about the `\\` (an exception to the rule).

4.9.6 Color in Mathematics

There is a temptation to use color to indicate or highlight portions of mathematics, especially for electronic outputs where color is easy and cheap. But before you leap, how will this work in black-and-white printed output? How will it work for a blind reader using a screen-reader or a braille version?

With careful use of TeX grouping (`{...}`) you can make the two behaviors of `\color` effective. For example, go:

```
{\color{blue}{x^2}}
```

4.9.7 Cross-References in Display Mathematics

A cross-reference is achieved with the `<xref>` element, see Section 3.4. You can place an `<xref>` inside a `<mrow>`, and remarkably, it will do the right thing. This is one of only two XML elements you can mix-in with LaTeX syntax. A typical use is to provide a justification or explanation for a step in a proof, derivation, or simplification. And it works best with alignment, see below.

4.9.8 Alignment in Display Mathematics

Displayed mathematics is implemented with the AMS-LaTeX `align` environment. Ampersands are used to control this, so use the `\amp` macro for these. The first ampersand in a line or row is an alignment point, typically a symbol, like an equality. The next ampersand is a column separator, then the next is an alignment point, then a column separator, then... The moral of the story is you should have n alignment points, with $n-1$ column separators, for a total of $2n-1$ ampersands—always an odd number.

For example,

```
<md>
  <mrow>A \amp = B \amp D \amp = E \amp \amp \text{Because}</mrow>
  <mrow>  \amp = C \amp   \amp = F \amp \amp <xref ref="txo" /></mrow>
</md>
```

produces

$$\begin{aligned} A &= B & D &= E & &\text{Because} \\ &= C & &= F & &\text{Table 4.5.3.} \end{aligned}$$

Sometimes you want several short equations on one line. Do not use `<me>`. Instead use a single `<mrow>` inside an `<md>`, and use alignment to spread them out evenly.

For multi-line display mathematics with no ampersands present, each line will be centered. This is implemented with the AMS-LaTeX `gather` environment.

You can fool the alignment behavior by hiding all your ampersands in macro definitions, so there is the optional `@alignment` attribute for the `<md>` or `<mdn>` element, in order to force the right kind of alignment. Allowable values are `gather`, `align`, and `alignat`. The latter is similar to `align`, but no space is automatically provided between columns. You can leave it that way, or explicitly add your own. For example, this allows you to precisely arrange individual terms of a system of linear equations, especially when terms with zero coefficients are omitted. When using the `alignat` option PreTeXt tries to count ampersands to see how many columns you intend, since LaTeX needs this number (we are not sure why). This detection can be fooled too, especially if you have something like a matrix with lots of ampersands for other purposes. So set the `@alignat-columns` attribute to the *number of intended columns*, if necessary.

4.9.9 Commutative Diagrams

Commutative diagrams may be authored using the AMS LaTeX `amscd`[4] package. While restricted in some ways, such as the lack of sloped or curved arrows, it has one important advantage over more general drawing tools. Support for HTML output comes from MathJax, and hence has accessible versions included in the output.

Typical use would be within an `<me>` element, so starting with `\begin{CD}`. Despite this being multi-line output, we have not chosen to integrate it within the more general `md/mrow` structure, but that decision can be revisited.

4.9.10 Fill-In Blanks in Mathematics

The other mix-in XML element is `<fillin>`. It may use a `@fill` attribute with some math string to determine width, height, and depth of the blank. Or it may use a `@characters` attribute that takes an integer value to hint at the width.

4.9.11 Page Breaks for Tall Display Mathematics

For print output, do nothing additional and LaTeX will do its best to break your display between lines. You can turn this behavior off by setting the `@break` attribute on the `<md>` or `<mdn>` to the value `no`. Once you do this, you can then selectively allow a page break after a given `<mrow>` by setting the `@break` attribute on the `<mrow>` to the value `yes`.

[4] `ctan.org/pkg/amscd`

4.9.12 Your Macros

These go in the `<docinfo>` section, wrapped in a `<macros>` element. Keep them simple—one or two arguments, and one-line definitions. This is not the place to be fancy, and not the place to try to end-run the structural aspects of PreTeXt. The idea is to define something like `\adjoint{A}` for the matrix `A` to be a superscript asterisk, and later you can change your mind and use a superscript dagger instead. Keep in the spirit of PreTeXt and use readable, semantic macros. For example, do not use `\a{A}` for the adjoint of `A`. Repeating: keep your LaTeX macros simmple, and to a single line.

PreTeXt will use your macros correctly for print and for HTML, after erasing whitespace from the left margin, and stripping LaTeX comments.

The name of your macros also cannot contain any numbers, otherwise MathJax will "silently" fail and may not read any subsequent macros you might have. This is important because PreTeXt will place custom macros for you at the end of your own, defined at Subsection 4.9.4, to be used. Those would fail to be processed by MathJax if your own macros caused it to stop reading.

4.9.13 Semantic Macros

We have resisted using overly-verbose MathML for mathematics, or worse, inventing our own XML vocabulary for mathematics. LaTeX syntax generally works great, but to work even better within PreTeXt an author needs to take a few extra steps. Your work will translate better to a variety of formats, and will be easier to maintain, with well-designed macros. A well-designed macro will convey the mathematical *meaning* of the object to a reader of your source, without them looking at your *definition* of the macro. In situations where a mathematical object might be written with different notation, it should be trivial to change the macro's definition and preserve the mathematical meaning. For example, consider two versions of a binomial coefficient:

$$\binom{n}{r} \qquad\qquad \mathrm{C}(n,r)$$

which could both equally well be the realization of `\binomial{n}{r}`.

Here we describe some notation which often carries multiple mathematical meanings and/or may be created with LaTeX in multiple ways.

4.9.13.1 Vertical Bars

Vertical bars are used for a variety of mathematical objects. Paired to create functions of expressions: absolute value, $|x-1|$; norm of a vector, $\|\mathbf{v}\|$; cardinality of a set, $|X|$; and the determinant of a matrix, $\left|A^k\right|$. As relations: division, $a \mid b$; parallel lines $L_1 \parallel L_2$. Sets are another use: $E = \{x \in \mathbb{Z} \,\middle\vert\, x \equiv 0 \pmod{2}\}$.

LaTeX `\vert`, `\Vert`, `\lvert`, `\rvert`, `\lVert`, `\rVert` are the delimiters, where `l` and `r` refer to left and right, and the capitalized versions are a pair of vertical lines. The qualifiers `\left` and `\right` can be used to have the length of the bar grow to match what it encloses. Note that there is a `\middle` that we have used above with `\vert` for the set E, and we have added space on either side. `\mid` and `\parallel` are relations, used above to indicate divisibility and parallel lines, and so automatically get an extra bit of spacing on either side.

When using `\left` or `\right` in isolation, `\left.` or `\right.` can be used to define a group that only has a bar on one end. For example:

$$\int_0^1 \sin^{-1}(x)\,dx = \left.\frac{1}{\sqrt{1-x^2}}\right|_{x=0}^{x=1}.$$

4.9.13.2 Times

The "times" symbol sees many uses that are different: dimensions, multiplication, and more complicated products (such as a Cartesian product of two sets). Macros `\product`, `\times`, and `by` could carry different meanings, even if each one is defined by the `\times` symbol, $\times$. For example:

```
Chess is played on an <m>8 \by 8</m> grid, which
contains <m>8 \times 8 = 64</m> little squares.

If <m>G</m> and <m>H</m> are finite groups, then
<m>\card(G \product H) = \card(G) \times \card(H)</m>.
```

4.9.14 Punctuation After Display Math

If a chunk of displayed math concludes a sentence, then the sentence-ending punctuation should appear at the conclusion of the display. (And certainly not at the start of the first line after the display!) But do not author the punctuation within the mathematics element, put it afterwards, where it logically belongs.

More specifically, place a sentence-ending period (say) *immediately* after the closing of an `<me>`, `<men>`, `<md>`, or `<mdn>` element. PreTeXt will place the period in your output in the right place and in the right way. (By using LaTeX's `\text{}` macro, if you are curious to know the details.) Here is an example. The XML source

```
<md>
    <mrow>(a+b)^2</mrow>
</md>.  Now...
```

will render as

$$(a+b)^2.$$

Now...

This all applies more generally to clause-ending punctuation, such as a comma. Take notice of the requirement that the punctuation must be *immediately* after the closing tag of the math element, otherwise it will not migrate properly. For example, do not interrupt the flow with whitespace, or an XML comment, or anything else.

For inline mathematics (the `<m>` element) the same authoring principle holds, though you would likely do this naturally. Author the punctuation *outside* the element, where it will remain.

Here is a technical subtlety that will demonstrate some of the inner machinery of PreTeXt and our conversions. In your work, locate a theorem that has some numbered display mathematics (`mdn`) which is at the end of a sentence, and which you have authored as described above. In HTML output, test a cross-reference (`xref`) to the theorem and you will see the period for the end of the sentence at the end of the display, where it should be. Now test a cross-reference (`xref`) to one of the numbered equations. First, the knowl will contain the entire display, to provide context, but it also will not contain the period, since the rest of the sentence is not in the knowl and so the period is not necessary.

Best Practice 4.9.2 Authoring Punctuation after Mathematics. *Always* follow the instructions in Subsection 4.9.14 about placing all punctuation following mathematics *after* the math element, not inside it. PreTeXt will do the right thing for display math for you. But furthermore, there are some special situations where the output format is not visual, such as braille or audio, where the placement of the punctuation is both different and important to not confuse "reader." You can help ensure your various outputs are of the highest quality by observing these sorts of details.

4.9.15 Lists of Mathematical Expressions

It is common to make lists of expressions, equations, or identities. Think of the definitions of trigonometric functions, a collection of antiderivatives, or a compendium of generating functions.

In these situations, author a list item, `<li>`, within an `<ol>` or `<ul>`, by using *only* the necessary `<m>` element. Do not use an intervening `<p>`, and do not include any adjacent text. Whitespace is OK. Then PreTeXt will add LaTeX's `\displaystyle` command to improve the visual appearance of the mathematics, and so you do not need to.

If you prefer to not have this behavior, insert an intervening `<p>`, and output will be identical, but without the `\displaystyle`.

Note that *any* text, other than whitespace, outside the `<m>` tag will disable this feature, *including punctuation*. However, according to the *Chicago Manual of Style* [1, 6.127], "Items carry no closing punctuation unless they consist of complete sentences." So that comma at the end of your equation probably doesn't belong there anyway.

4.9.16 LaTeX Packages, MathJax Extensions

LaTeX, which we use as the syntax for mathematics, allows extensive customization through **packages**, which are enabled through a `\usepackage{}` macro. Packages also provide extensive customization or control over the document structure. Thus, PreTeXt provides support for additional packages *to enhance mathematical content*, but does not have any facility beyond this (e.g. to influence or support document structure).

We use LaTeX to create PDF and print output. For most other electronic formats, we use MathJax to embed mathematics onto a web page or to make images of mathematics used in EPUB, and so on. MathJax has **extensions**, which mimic certain LaTeX packages. For a list of supported extensions see MathJax's TeX/LaTeX Extension List[5]. Note that some of these extensions are technical (not supporting mathematical syntax) and some we load already as part of setting up MathJax, so carefully evaluate how a given package is selected to support certain macros.

Suppose there is a LaTeX math-mode macro you would like to use. We will illustrate with the `\cancelto` macro that is part of the fairly standard `cancel`[6] package. It will draw an arrow through a mathematical formula, with a replacement value. Conveniently, MathJax has a `cancel` extension, and this extension *also* defines the `\cancelto` macro. You need to check this, as a LaTeX package and a MathJax extension are not guaranteed to define the same collection of macros. Nor are the two guaranteed to have the same name (though this is best practice).

To make the `\cancelto` macro available and active in your mathematical elements (`<m>`, `<me>`, etc.) add the following to your `<docinfo>` element:

```
<math-package latex-name="cancel" mathjax-name="cancel"/>
```

Now you may freely use the `\cancelto` macro in your mathematics, confident that it will render correctly in all output formats.

Note that both attributes `@latex-name` and `@mathjax-name` must be present. They may be identical (as above, and typical) or they may be different (rare). There are situations where one might be empty. Of course, you can use these macros in other definitions of mathematics macros you might make (Subsection 4.9.12). However, we do not guarantee the absence of conflicts with other packages in use, even if employed by PreTeXt. Nor do we support debugging such conflicts.

If there are LaTeX macros you desire, but without a MathJax implementation, then there is not much we can do, short of suggesting you write your own extension (not recommended, see A Custom Extension[7]). There are some workarounds which we may document shortly.

4.9.17 Mathematics in Titles

Titles are a key feature of PreTeXt, and migrate to various places in different conversions, where they might be mixed with bold text, or where they do not render properly. Please report any substandard situations. A good example of this phenomenon is mathematics in division titles, where we provide a `<plaintitle>` alternate. See Section 4.8 for a complete discussion.

4.9.18 Extras

We were once in the business of supporting "extra" macros for mathematics, but have discontinued that practice (in a mostly backward-compatible way).

We are moving away from built-in automatic support for the `\sfrac{}{}` macro. As of 2023-10-18 you may use this macro, but results will just look like regular inline fractions. However, you can get superior typesetting in PDF output by loading the `xfrac` LaTeX package with the following in `<docinfo>`:

```
<math-package latex-name="xfrac" mathjax-name=""/>
```

See the Subsection 4.9.16 for more details.

Historically, we provided good internal support for the LaTeX `extpfeil` package of extensible arrows, even though the LaTeX version has some severe shortcomings. As of 2023-10-19, it is now an author's responsibility to *elect* this package, along with the reliable MathJax extension of the same name. Add to `<docinfo>`:

[5] docs.mathjax.org/en/latest/input/tex/extensions/index.html
[6] ctan.org/pkg/cancel
[7] docs.mathjax.org/en/latest/web/webpack.html#custom-extension

```
<math-package latex-name="extpfeil" mathjax-name="extpfeil"/>
```

See the Subsection 4.9.16 for more details.

4.9.19 Notes

As mentioned at the start of this section, your use of LaTeX needs to also be supported by MathJax so that it may be rendered as part of an HTML page displayed in a web broswer. In addition to the information at the start of Section 4.9, this subsection has some notes that may help you navigate this situation.

1. Generally, MathJax supports commands available in the `amsmath` package.
2. You can construct, and use, your own macros, *but only for mathematics*, not for document structure or document management. See Subsection 4.9.12.
3. Support for loading "extra" packages is extremely limited. See Subsection 4.9.16.
4. The `\matrix{}` command, and its friends (such as `\pmatrix{}`) are not supported by LaTeX, despite being recognized by MathJax. So use environments like `\begin{matrix}` and `\begin{pmatrix}` (with their corresponding `\end{}`, of course) and you will get accurate results for both formats.

4.10 Mathematics Best Practices

This section addresses some finer points of authoring mathematics with LaTeX, motivated in large part by helping MathJax create the most accessible output possible, which in many instances will also create a superior typographical result across all conversions. We try to provide justification and explanation, though that might not be easy in all cases. Some are definitely technical, and we are only aware of them from authors' experience and reports. Many are meant to help in the conversion to braille. This is an incomplete list so additional reports and additions are welcome.

Large Numbers. Using a separator to aid in reading a large number should avoid using a space as a separator. So $234{,}766{,}544$ or $234.766.544$ is preferable to $234\,766\,544$. We have used braces around the commas in the first instance, and a LaTeX "thin space", `\,` , in the last instance.) For $234{,}766{,}544$, braces around each comma prevents automatic trailing space that would normally be desirable in a list of numbers. In other words, author as:

```
<m>234{,}766{,}544</m>
```

Here is the version with commas and no effort to distinguish the big number from a list of smaller numbers: $234, 766, 544$.

Ratios. If you really intend to communicate a ratio with a colon, keep it super-simple, like `3:2` or `a:b`. Do not wrap it in parentheses or other decorations, since then the colon will be communicated literally in braille. For example, the index of a subgroup, `[G:H]`, will not be confused with a ratio, due to the brackets. But `(4:5)` will not be output in a conversion to braille in a way that communicates that it is a ratio. If you keep your ratios simple, braille output will use special syntax designed for ratios.

Function and Operator Names. Common mathematical functions, like sin (`\sin`) and det (`\det`), are built into LaTeX, well-known, and commonly used. Compare with the versions authored as simple sequences of letters sin and det, which LaTeX interprets (and typesets) as a product of three individual variables.

Instead, use the LaTeX `\DeclareMathOperator` macro as we will illustrate. Suppose you want to discuss the set of homomorphisms from the vector space U to the vector space V, $\operatorname{Hom}\left(U,\,V\right)$. Define the macros

```
\DeclareMathOperator{\homop}{Hom}
\newcommand{\Hom}[2]{\homop\left(#1,\,#2\right)}
```

and employ as \Hom{U}{V}, which yields Hom $(U,\ V)$. Compare with the no-macro, no-special-care, version, $Hom(U,V)$. Grouping Hom as a unit will prevent LaTeX from rendering it as a product of three variables, use the correct font, and will preserve the meaning in Nemeth braille. Notice that the \homop macro is never used in your source.

Directly contradicting this general advice, Sean Fitzpatrick reports on 2021-01-06 that the LaTeX macro construction

```
\newcommand{\foo}{\operatorname{foo}}
```

behaves better in Asymptote diagrams than

```
\DeclareMathOperator{\foo}{foo}
```

So perhaps you need to define a second macro for use in Asymptote diagrams.

Permutations in Cycle Notation. The permutation in cycle notation, $(1246)(35)$, is difficult to distinguish from a product of two integers and gets entirely different treatment than intended as Nemeth braille. And when the points of the permutation group involve multiple digits, some other notation will become necessary anyway. Commas work, as in $(1,2,4,6)(3,5)$. Or if commas look too cluttered, then spaces are possible, as in $(1\;2\;4\;6)(3\;5)$, where we have used the LaTeX medium space, `\;` .

Text in Math. We have discussed this one already, but it is important for a conversion to braille. If you use a word inside of math, such as describing a condition for membership in a set, put any spaces inside of a `\text{}` macro, not on either side of it. So `\text{ and }` is preferable to `\ \text{and}\ `. Similarly, do not put any math inside a `\text{}` macro. For example,

```
\text{ for -3 \lt  x \lt  -2 }
```

will lead to poor results due to the inequalties mixed in with the text. A better version would be

```
\text{ for }-3 \lt  x \lt  -2
```

and now the numbers will be treated as mathematics. Review Subsection 4.9.5 for more details.

Math as Text. Do not use symbols from mathematics in non-mathematical situations such as the LaTeX construction `73^\circ` to indicate a temperature in degrees, outside of a mathematical or scientific setting, such as in a <poem> about springtime weather. In this particular case, we provide a <degree/> element that can be used in non-mathematical contexts, so we can get 73°, which will behave well in a variety of conversions (and can also be used for latitude and longitude in your poem). Similarly, we advocate writing ordinal numbers with text on the baseline, rather than a LaTeX construction like `2^\mathrm{nd}`. So use 1st, 2nd, 3rd, 4th, etc.

Avoid Nesting. It may seem convenient to nest custom macro definitions. For example:

```
\newcommand{\makered}[1]{{\color{red}{#1}}}
\newcommand{\MAKERED}[1]{\textbf{\makered{#1}}}
```

The second macro definition uses the first macro. At least one PreTeXt conversion isolates macro defintions for use only on an as-needed basis. So here, if the author used `\MAKERED` without also using `\makered` close nearby, it would lead to an issue under that conversion, because `\makered` would be locally undefined. So we recommend defining each macro from the ground up. In this case:

```
\newcommand{\MAKERED}[1]{\textbf{\color{red}{#1}}}
```

This also contradicts our advice above on operator names.

4.11 Lists

A **list** is an unusual construction, even if everybody knows exactly what one is. We view the list itself as a container of various chunks of text, while those chunks of text are the **list items**. Each item has a **marker** to identify it.

Markup and processing is complicated by the possibility of a list item containing a list, resulting in **nested lists**. We simplify this problem by *requiring* that a list appear within a paragraph (`<p>`), see Subsection 4.1.3. One of the three exceptions is the possibility to place a list into a block that earns a caption and a number, using the `<list>` element, see Section 4.20.

The final subsection contains some examples that you may wish to consult as you read this section.

4.11.1 Ordered, Unordered, Description Lists

An **ordered list** has items with markers that are naturally ordered (usually numerically or alphabetically). We borrow from HTML, and use the `<ol>` tag to construct an ordered list. Some commentators suggest an ordered list should only be used when the order of the content is important. So the steps in a recipe would belong in an ordered list, but the shopping list when you go to the store need not be an ordered list.

An **unordered list** has items with markers that have no inherent order and so are usually symbols like circles, disks, squares, etc. We borrow from HTML, and use the `<ul>` tag to construct an unordered list.

A **description** list has items that have *short* pieces of text as their markers. We borrow from HTML, and use the `<dl>` tag to construct a description list.

Ordered lists are used as part of `<objectives>` (`[provisional cross-reference: topic-objectives]`) and exercises (Section 4.13). Any of the three lists may occur inside the `<list>` element (below, Section 4.20). Otherwise, a list must occur within a paragraph, `<p>`. This means that to place a list within a list item of another list, the list item must contain a paragraph.

4.11.2 List Markers

Do nothing, and your ordered and unordered lists will get sensible default markers. They are consistent in the following sense. If your list has two items, and each of the two items contains a list, then these two lists will use the same type of marker.

For a description list, you author each marker as part of each list item, as discussed below in the discussion of list items.

If you want to change how an ordered list is marked, then you use the `@marker` attribute on the `<ol>`, whose value is a **format code**. This string contains one of five codes (a single character), which may be surrounded by other characters, excluding the five codes. For example `marker="(A)"` will produce uppercase letters wrapped in parentheses: (A), (B), (C), ... The extra formatting works well in a conversion to LaTeX, but is not possible technically in a conversion to HTML, so it should be considered decorative, and not relied upon for meaning. The formatting does not carry through to the numbers of list items in cross-references.

If you want to change how an unordered list is marked, then you use the `@marker` attribute on the `<ul>`, whose value is a **format code**. This string contains one of three codes in the table below. Then every item of the list will have that symbol as its marker.

Code	Realization
1	Arabic numerals
a	Lowercase letters
i	Lowercase Roman numerals
A	Uppercase letters
I	Uppercase Roman numerals
0	Arabic numerals, from zero

Table 4.11.1 Ordered List Markers

Code	Realization
disc	Filled small circle, aka a bullet
circle	Small circle
square	A square

Table 4.11.2 Unordered List Markers

Default marker types are assigned to each level of nested lists in the order shown in the table, and cycle back to the top of the table if necessary, though zero-based Arabic numerals will be skipped for ordered lists.

Start with the defaults, and experiment as needed. See the examples below for some extreme (and unwise) customizations of markers.

For a description list, possible markers are more varied than what you can express within an attribute. So the list item must have a `<title>` element (see below). This should be very short text, and may contain inline mathematics. It is often rendered in bold, so be aware that some markup may get lost. Perhaps for obvious reasons, do not include footnotes, cross-references, or display mathematics. The `<dl>` element has a `@width` attribute, with possible values `narrow`, `medium`, and `wide`. The default is `narrow`. This is a *hint* about how much text you have in these markers, and in certain presentations may make better use of horizontal space on a page.

4.11.3 List Items

So now you have a list all organized as a container. What do you put in it? List items, using the `<li>` tag, again borrowed from HTML, and independent of the type of list.

A list item could be really simple, maybe just one or two words. Then you can use, and conceptualize, an `<li>` element as not much different from a `<p>` element, and the rules about content are not much different. Even several full sentences, with some intermediate displayed mathematics, is fine.

But once you want two paragraphs in a list item, then you need to structure the contents of the item. So a list item might have five paragraphs in it, requiring five `<p>` elements. Notice that this is how you nest lists. Make a list item, include a paragraph, then put the subsidiary list into the paragraph. Indeed, this is the only way to nest lists. A consequence of this is that the only way to have an unstructured list item is if it is a terminal item, like the leaf of a tree.

Other items may be interspersed among the paragraphs of a list item, such as a chunk of verbatim text delimited by a `<pre>` tag. But anything with a number, such as a `<figure>` or `<remark>` is banned, in part because the consequences for numbering and organization become too complicated. Imagine a remark, and a paragraph of the remark has a list. Fine so far. But if the items of that list can again contain remarks, the possibilities become endless. You should be able to include non-textual items, like an `<image>`, and work is underway to improve this. You can use a `<sidebyside>` in a structured list item, which can in turn hold an `<image>`, `<tabular>`, or similar. But you cannot place items in such a `<sidebyside>` that are numbered, so a `<figure>` or `<table>` is not possible. A general rule is no numbered components in a list item. Computational components, such as `<sage>` are also banned from list items due to the difficulty of converting them into electronic computational notebooks with a relatively flat structure.

A list item of a description list must have a `<title>` element, to provide the text of the marker. Now that the list item has some structure, the remainder must also be structured, typically with some paragraphs, as discussed above. In other words, the earlier option of employing an `<li>` element just like a `<p>` element is not available in a description list. Further, given the complexity of presenting a description list, it can *only be a top-level list.* It can contain the two other types nested within its list items.

For ordered and unordered lists, you may optionally include a `<title>` when you have structured the `<li>`. This will be rendered as a heading of sorts for the list item, though the only distinction might be a change to an italic or oblique font. As an example, this might be a good way to author a Frequently Asked Questions (FAQ).

Note that a list item holding *only* an `<m>` element will get special treatment. See Subsection 4.9.15.

4.11.4 Cross-References to List Content

Note that a list is a container, so it cannot be the target of a cross-reference, and so the three types of lists cannot have an `@xml:id` attribute. But you may well be able to point at some other structure (e.g., a `<remark>`) with a paragraph containing a list of interest. If this seems overly restrictive, read below about named lists.

By contrast, a list item, `<li>`, is not a container, and does contain content. Further, a list item of an ordered list has a marker that is natural text for a cross-reference. So in this situation, the list item can have an `@xml:id` attribute. But note that the "number" of a list item of an ordered list, which is nested inside a list item of an unordered list, is not defined, so a cross-reference by number can fail.

The "number" of a list item, mostly for the purposes of a cross-reference, is the concatenation of all of the individual markers in the containing lists, outermost first. For example, from the example lists below, the list item with content "Walleye" has number 2.I. These are indivisible, there is no way to get a component, excepting leading subsequences obtained by using an `@xml:id` on a containing list item. Note that the format codes never become part of the number.

4.11.5 Lists in Columns

You can control the number of columns used to layout an ordered or unordered list (but not a description list). On the `<ol>` or `<ul>` use a `@cols` attribute with values `2` through `6`. (`1` is the default.)

We do not yet (2018-03-28) have enough technical confidence to allow an author to specify a row-major order versus a column-major order for the layout. So understand that this is can be an implementation choice for a particular conversion, and can vary across implementations. If this is critical to conveying *meaning*, and not an aesthetic preference, then maybe consider using a `<table>` or `<tabular>` (Section 4.18).

Best Practice 4.11.3 Use Only a Few Columns for Lists. Anything more than three columns tends to get very crowded horizontally. Think twice about using more than that, and realize that six columns should be a ridiculously generous upper limit, and not a promise of good behavior in final output.

4.11.6 Exceptional Lists

We use the tags for lists in a few situations outside of anonymous lists inside paragraphs and named lists. These include the items within an objectives, subparts of an exercise, and within panels of a side-by-side. See those topics to learn about subtle differences in use.

4.11.7 Examples of Lists

To illustrate this section, we offer three too-elaborate examples. Take these as compact examples of what is possible, and not best practice in your writing. We also use these to illustrate cross-references to list items, see Subsection 4.5.6.

We have a paragraph that begins with anonymous list of species that live in water (maybe partially), which necessarily is placed inside a paragraph. The roman numerals purposely do not have any extra adornment in the LaTeX version (but may for HTML output).

1) Amphibians
    a. Frog
    b. Salamander
    c. Newt
    d. Toad
2) Freshwater Sport Fish
    I Walleye
    II Bass
    III *Exquisite Fish.*
    Trout
3) Saltwater Sport Fish
    (A) Salmon
    (B) Halibut
    (C) Marlin

Within the same paragraph, we transition to an unordered, two-column, list of some germs:

- Bacteria
    - *Staphylococci*
    - *Streptococci*
    - *Salmonella*
- Viruses
    - *Varicellovirus*
    - *Orthopoxvirus*

This sentence concludes our (small) paragraph on small and large organisms.

A named list, only to test cross-references.

List 4.11.4 A two-deep ordered list

- A:
    - i. A and i
    - ii. A and ii
- B:
    - (a) B and a
    - (b) B and b
    - (c) *A Test Title.* B and c
- C:
    - <I> C and I
    - <II> C and II
    - <III> C and III

An example of a description list, anonymously in a paragraph.

Red	The color of the sun at sunset.
Blue	The color of a clear sky.
Aqua	The color of shallow tropical waters.
$x^2 + y^2$	Definitely not a color

Remark 4.11.5 Best Practice. Lists are a very attractive device. Hopefully the discussion above has convinced you that they are more complicated than they first appear. Think carefully before using one, and consider if some other structure (`<paragraphs>`, `<sidebyside>`, a subdivision) might do a better job of organizing and communicating your meaning. And if a list is really necessary, consider if it should be named or anonymous, heavily-nested or nearly-flat, with columns, or with long or short content in the items. Cross-references *from* the items of a list *to* more complicated structures is another device that works well.

4.12 Interactive Exercises

PreTeXt has markup for a variety of interactive problem types, described individually in the subsections to follow. Generally, for a regular HTML build (Chapter **??**), these will be interactive, informing the reader when their answer is correct, and providing feedback when their answer is not correct. When HTML is built to host on a Runestone server (Chapter **??**), then student progress as part of a course will be recorded on the server. Some features, which will be noted below, such as execution of computer code, is more capable on a Runestone server.

Generally, an interactive exercise is authored much like a “regular” `<exercise>` (Section 3.9) with `<statement>`, `<hint>`, `<answer>`, `<solution>`. However, there is also some additional markup which serves as a **signal** that the exercise is more than a short-answer, or free-response, question. Usually, but not always, this signal is an additional element following `<statement>`. One consequence of this is that all but one type of interactive exercise will require a `<statement>` element to contain the question text.

Each type of interactive question has a **static** version for use in output formats like print, PDF, or braille. Details are given below. Note that since the ability for instant evaluation of a reader’s answer means an author provides the solution and other feedback, this can then be incorporated into a static version as an automatically-generated `<answer>` and/or `<solution>` (in addition to any others an author provides). Note that an author can provide a `<hint>` for use in all output formats, but there are not any automatically-generated `<hint>` for static versions of interactive problems. Visibility of these solutions can be controlled via the mechanism applicable to all exercises, see Section **??**.

Note the opportunity to provide feeback to the reader using a `<feedback>` element in various locations. These are generally optional, but highly encouraged as a way to improve the quality of your reader's experience.

These interactive questions are enabled by software from Runestone Academy (Chapter ??) and have extra capabilities when the online version of our output is hosted on Runestone servers. But you *must* use a `@label` attribute. This is described more carefully next, so that we can make reference to it from other locations.

Best Practice 4.12.1 Use `label` attribute for Runestone Components. Many of our interactive exercises and programming environments are powered by software from Runestone Academy (Chapter ??). Not suprisingly, when you build online HTML for hosting on Runestone servers these components have more features than otherwise. This is managed by locating each component in a (massive) database. And so each exercise needs an identifier.

You accomplish this by placing a unique value in a `@label` attribute on an outermost element, such as an `<exercise>`, `<task>`, `<project>`, `<video>`, `<program>`, etc. You should choose these strings carefully, as they should not be changed, lest the database entries become confused. Under the hood, we get the content of `docinfo/document-id` element, and the `@edition` attribute of `<document-id>`, to form a database identifier such as `AATA_2_easy-exercise`. The `<document-id>` distinguishes your book from other books, and the edition allows you some flexibility. In other words, if you declare a new edition, you can change some of your `@label` as part of that process and you will get new database entries while preserving the old.

Short answer: *before* hosting your project on Runestone, decide on permanent values for `<document-id>`, `@edition`, and all necessary `@label`. (We expect to add a warning for Runestone builds when these values are not set.)

4.12.1 True/False Exercises

A True/False exercise *must* have a `<statement>` element, and this element *must* have a `@correct` attribute, whose value is `yes` or `no` (there is no default value). That's it. The presence of the `@correct` attribute is the signal that this is a True/False exercise.

The text of the statement is the assertion the reader must determine is true or not. The `@correct` attribute is how an author describes if the statment is true (`yes`) or false (`no`). This is enough information for a conversion to formulate a version of the question. An optional `<feedback>` element may follow the `<statement>`, and should provide more thatn a binary explanation of the exercise.

Presentation as an interactive element will vary cosmetically, according to the output type targeted.

A static version gets an automatic `<answer>` that is simply "True" or "False". The automatic `<solution>` is the same, plus the content of `<feedback>`.

4.12.2 Multiple-Choice Exercises

A multiple-choice exercise is signaled by a `<choices>` element (plural) following a `<statement>` and preceding an optional `<hint>`. The `<choices>` element is structured by a sequence of `<choice>` elements, whose content is a potential answer for the reader to choose. So the `<statement>` is the prompt or question, and the `<choice>` are the possible answers.

At least one `<choice>` has an attribute `@correct` set to the value `yes`. The default value of `@correct` is `no`. There may be several correct answers, indicated with this attribute. The presentation as an exercise with one answer or many is automatic. However, in the event there is exactly one correct answer, but you wish the reader to consider the possibility of multiple correct answers, you may set the `@multiple-correct` attribute on the `<choices>` element to `yes`. The default value is `no`.

Each `<choice>` element must be further structured with a `<statement>` and a `<feedback>`, which each can contain items such as paragraphs (`<p>`). In this way, the highly-encouraged feedback can be associated with each correct and incorrect choice.

The order of the `<choice>` as authored, is the order they will be given in a static version. To present the choices in different orders in an interactive version, set the `@randomize` attribute on the `<choices>` element to `yes`.

An automatic `<answer>` for the static version is simply the list markers for the correct choices. An automatic `<solution>` has an indication for each choice if it is correct or incorrect, along with the choice's feedback.

4.12.3 Parsons Problems

Parsons problems are named for Dale Parsons, one of their creators, along with Patricia Haden. They could also be called **mixed-up blocks**. A reader is presented with a set of blocks containing text, either computer code or natural language, and their goal is to assemble the blocks into a correct order. This could be a computer problem with a stated purpose, or could be a logical argument such as proof, or it could be a procedure such as a recipe. An interactive drag-and-drip interface is a very efficient presentation for a reader.

Similar to multiple-choice exercises, a `<statement>` is followed by a `<blocks>` element containing a sequence of `<block>` elements. The `<blocks>` element is the signal that this is a Parsons problem. The overall `<exercise>` element may have several attributes:

- `@numbered` set to `left` or `right` indicates the blocks should be numbered, and on which side the numbers go. The default value is `no`.
- `@language` set to `natural` indicates the text of the blocks is natural language, while the use of a computer language should be indicated by naming the actual language employed. A list of languages will soon be added in Section 4.15. The default is `natural`.
- `@indentation` set to `hide` indicates that a computer code exercise will not include indentation common to each line of a block. In other words, the problem is slightly harder, as the interactive interface will require the reader to adjust the block horizontally to provide common indentation for the block. The default is `show`.

A `<block>` that has natural content is authored as usual, with a sequence of paragraphs (`<p>`) or similar. If the content is a computer language, then it should be authored as a sequence of lines, using the `<cline>` element, which allows for precise interpretation of indentation and non-standard characters. Include *all* indentation necessary for each line, no matter how `@indentation` is set. The content of the blocks, arranged properly, should form a syntactically correct program. If the reader is to provide indentation, PreTeXt will strip away the common amount of indentation.

A block may be a **distractor**, meaning it is not to be used at all in the solution. Indicate this with a `@correct` attribute on the `<block>` set to `no` (the default being `yes`). Furthermore, a given block of the solution can be authored with several alternatives, only one of which is correct. Indicate this by structuring a `<block>` with a sequence of `<choice>`. Each `<choice>` should be authored similarly to a `<block>`, and the one correct choice should have the `<correct>` attribute set to `yes` (the default being `no`).

The blocks should be authored in the correct order and the interactive interface will control the randomization. A block that is a distractor may be placed in any location amidst the other blocks. Each block should have an `<@order>` attribute that is a whole number, counting from 1. This is the fixed, mixed-up, order that will be presented in any static rendition. In static versions, sequences of blocks are in lists, which are numbered if the attribute has been set, but the left/right distinction is lost—all numbers are to the left. An automatic `<answer>` is provided if blocks are numbered, and it is just the block numbers in the correct order. An automatic `<solution>` is always generated with the text of the blocks listed in the correct order. For an exercise with computer code, the PreTeXt `<program>` element and the exercise's `@language` attribute will produce a syntax-highlighted listing.

There is no provision for a `<feedback>` element. Further explanation may be provided in a `<hint>`, `<answer>`, or `<solution>`. Also, a good interactive interface can provide assistance to a reader by telling them if they have too few or too many blocks, or combining or removing blocks, and so on.

As of 2022-06-14 work is underway to relax the strict ordering assumed of the blocks. So there will be extra markup to specify a poset or directed acyclic graph on the blocks.

4.12.4 Horizontal Parsons Problems

A **horizontal Parsons problem** is very similar to the traditional *vertical* problems just described in Subsection 4.12.3. Except they are horizontal. Which means the blocks are very short (a word or symbol) and they are rearranged by the reader to form a horizontal bit of text. They are ideal for constructing a single line of computer code, such as a regular expression, or a short sentence. While similar, they have a few features which are different.

To indicate a horizontal problem, set the `@layout` attribute on the `<blocks>` element to the value `horizontal`. The default value is `vertical` and so is not necessary for the traditional versions.

The syntax for blocks is the same, and they are authored in an order that yields a correct answer. The use of an `@order` attribute on each `<block>` indicates a rearrangment to be used for a static version. Blocks may be reused. To do this, place an `@xml:id` on a single instance of a block to be reused. Then, to indicate its reuse as part of a correct version, use a `@ref` attribute on a `<block>`. Since the duplicate instance is not shown to the reader, the use of `@order` and `@ref` is mutually-exclusive. You can set the `@reuse` attribute on the `@blocks` element to the value `yes` to force the interface to allow the reader to reuse blocks, even if there is no duplication in the correct version. Otherwise, the interface reacts to the presence or absence of blocks with the `@ref` attribute.

Distractors may be included in the same manner as for vertical Parsons problems, by setting the `@correct` attribute on a `<block>` to the value `no`. The default is `yes`. It does not make much sense to indicate a reusable block (in either form) as a distractor, so don't.

Another option on the `@blocks` element is the `@randomize` attribute. The default value is `yes`. When the value is set to `no`, then the blocks are always presented to the reader in the same order, which is the one given by the `@order` attribute.

As of 2022-11-28, the implemenation of these problems assumes that the blocks rearrange to be computer code. So you *can* set the `@language` attribute on the `<exercise>` element to `natural`, but it will not have much effect. Markup like `<em>` on the text of a block will be unpredictable, as will attempts to use mathematics. And the text of blocks will always be in a monospaced font, perhaps with some syntax highlighting if `@language` is set properly.

4.12.5 Matching Exercises

A **matching exercise** asks a reader to pair a **premise** with a **response**. Similar to multiple-choice exercises and Parsons problems, a `<matches>` element follows a `<statement>` and this is the signal. The `<matches>` element is structured as a sequence of `<match>` elements, each of which has a `<premise>` element and the matching `<response>` element. Since the content of each premise and response is best kept short and simple as a phrase, the elements may also be simple phrases without the additional structure of `<p>` elements, or similar. That's it.

An interactive interface should randomize at least one of the lists of premises and responses, consulting the authored version for the correct pairings. For a static version, an author should put an `<order>` element on each `<match>` whose value is a whole number, starting from 1. Then the `<premise>` will appear in the authored order, while the `<response>` will be re-ordered according to the attribute.

A single `<feedback>` element may be given, as a peer of `<statement>`, in addition to authored `<hint>`, `<answer>`, or `<solution>`. For a static version an automatic `<solution>` presents the problem in the order the `<match>` were authored.

4.12.6 Clickable Area Exercises

Clickable Area is a misnomer, assuming an interactive version of this type of problem. Perhaps "Select Text" would be better. In any event "area" is not meant to connote a geometric notion, instead this is about a sequence of characters, either in natural language or in computer code.

A `<statement>` is a prompt, describing which sort of areas should be selected by the reader. Such as: Locate all the nouns in the following paragraph. This is followed by an `<areas>` element, which is the signal for this type of exercise. The `<areas>` element holds either (a) PreTeXt paragraphs or similar, or (b) a sequence of `<cline>`, understood to be program code. In the latter case, a `@language` attribute on the `<areas>` element will enable the right syntax highlighting in some output formats.

Within the content of the `<areas>` element an `<area>` element can be used to surround a run of characters. These cannot cross XML boundaries, such as a `<cline>` or `<p>`. A `@correct` attribute, with values `yes` or `no`, indicates the text is to be selected (clicked on) or that it is a distractor (tempting to click on). The default is `yes`, so only distractors need to be indicated.

A single overall `<feedback>` may be placed following the `<areas>` element.

Now an interactive version will allow the reader to click on and off the marked areas of text, and provide information about which are correct and which are not, in addition to general feedback.

Generally, a static version of the problem will not be clickable. `<wink/>` So your prompt might say something generic like "select" or "locate" since a reader might work the problem on paper by circling or underlining the areas. A static version includes an automatic `<answer>` which is a list of the correct areas as text, followed by a list of the incorrect areas as text. An automatic `<solution>` repeats the text in the `<areas>` element, and uses (accessible) visual cues to note the correct and incorrect text.

4.12.7 Fill-in-the-Blanks Exercises

As of 2022-06-14 a major effort is underway to provide comprehensive markup for **fill-in-the-blank** problems. Until then, there is transitional markup intended only to supply a migration path for projects originally authored for Runestone servers (Chapter ??). So (a) our documentation is sparse, and (b) there will be no backward-compatible improvements. So in particular, new projects should wait for the new markup. Also, studying examples may be a useful way to augment what is described here.

A `<statement>` is enriched with empty `<var/>` elements which will render as the blanks in the problem.

The signal for a fill-in problem is a `<setup>` element containing a sequence of `<var>` elements. Each `<var>` contains a sequence of `<condition>` elements that describe possible values (via regular expressions) which might appear in a blank. The first condition describes the correct answer(s), and then the subsequent conditions are descriptions of probable incorrect answers. Each `<condition>` has a `<feedback>`. So the first condition to match an entry provided via a blank will be noted as correct or incorrect, and its feedback will be relayed.

The `<var>` of the statement and the `<var>` of the setup are in a 1-1 correspondence, which establishes how the setup is associated with a blank. The `<var>` in the statement may have a `@width` attribute whose value controls how many characters would be visible in the blank.

A static version will include an automatic `<solution>` which "fills in" each blank with a correct answer, and then duplicates the feedback text, in order.

4.12.8 Coding Exercises and Projects

A **Coding Exercise** is formed by placing a `<program>` element *after* the `<statement>` of an `<exercise>`. The main distinction is that this is a signal that the interactivity is provided by the `<program>`, and therefore the `<exercise>` will not be understood as a short answer exercise (Subsection 4.12.9). For this reason, the `<program>` should be requested as one of the interactive realizations (Subsection 4.15.2, Subsection 4.15.3).

The identical construction may be used with any PROJECT-LIKE (List 4.2.2) such as an `<activity>`. As of 2022-06-17 this only applies to the form that uses a `<statement>`, but will soon also apply to `<task>` within PROJECT-LIKE.

Realize that it is always possible to place a `<program>` *inside* of a `<statement>`, and if there is no `<program>` that is *after* the `<statement>` (or another signal for an interactive exercise) then the `<exercise>` will be classified as a short-answer exercise. It may be instructive to understand that in a static realization, a `<program>` at the *end* of a `<statement>` may be visually identical to an `<exercise>` where the `<program>` is *after* the `<statement>`, even though the former is a short-answer exercise and the latter is a coding exercise (which will render differently for different output formats and hosting platforms).

4.12.9 Short Answer Exercises

Short Answer questions might also be known as **Free Response** questions, or **Essay** questions. A PreTeXt `<exercise>`, or a PROJECT-LIKE that is not structured by `<task>`, is implicitly of this nature. But you still need to signal that you wish such a problem to be interactive, typically with a text box where the student can enter an answer. So, similar to other types of exercises, add a `<response/>` element immediately after the `<statement>`. (We expect to add attributes for this element to influence the behavior of the text box.)

In an online setting, it is a simple matter to provide a place for a reader to type in an answer, response, or essay. But then what? Until artificial intelligence is brought to bear, somebody (not something), such as an instructor for a course of enrolled students, will need to read a response and provide a score and/or comments that can be saved and distributed back to the students. So the first prerequisite is that HTML output is being built for a capable platform. As of 2022-06-15, this means a Runestone server (Chapter ??), but it could easily also be a WeBWorK server.

A publisher can control when a response area is created in HTML output (Subsection ??). The default is to only have this area present when it is possible for a response to be graded and scored. However, an option will cause a response box to be created always. A reader can reflect on the question by typing in a response, and the text will be saved *on that particular device* only. When it is impossible for a response to be graded, placeholder text will warn the reader.

4.13 Exercises and their Solutions

As described in Section 3.9 an `<exercise>` can be placed in many different locations, and a `<project>` has similar features. It is critical to understand that you want to author any hints, answers, or solutions immediately following the statement of an exercise. If your PreTeXt source is public, and you would like to keep some aspects of the solutions private, then read Section **??** for some practical advice. See Section **??** as well for information on creating a standalone *Solutions Guide*. We concentrate here on techniques for controlling visibility and location of the components of exercises within your primary output.

4.13.1 Exercises, Original Versions

In a conversion to HTML, a `<hint>` to an `<exercise>` renders nicely in a knowl, right below the exercise statement. For a conversion to LaTeX/PDF/print, you might wish to display a hint, visibly, as part of the exercise, or you may wish to park the hint in a *Hints to Exercises* division in the back matter. To control visibility of the components of exercises (and projects) there are twenty switches you can use. See Section **??** for more.

4.13.2 Exercises, Solutions Versions

Exercises, and their components may be duplicated easily, to provide a back matter appendix with solutions, or within each division. For example, you can easily create an end-of-chapter division with solutions to every inline exercise throughout the chapter and solutions to all the divisional exercises from each section of the chapter.

The `<solutions>` element will create an entire division, semi-automatically. You can provide a `<title>`, an `<introduction>`, and `<conclusion>`. The remaining content is statements, hints, answers, and solutions to exercises (and projects).

If `<solutions>` is a child of `<backmatter>`, then an appendix will be generated, and covering `<exercise>` from the entire `<book>` or `<article>`. If `<solutions>` is a child of a division, then a new subdivision is created and the scope is all `<exercise>` for the division. So, for example, a `<solutions>` placed inside a `<chapter>` will render as a division that looks like a `<section>` and will include components of all the exercises (at any level) contained within the `<chapter>`.

An optional attribute is `@scope`, whose value is the `@xml:id` of a division. Then it is this division which is scanned for exercises and their solutions (rather than defaulting to the enclosing parent of the `<solutions>`). This allows for much greater flexibility. For a simple example, suppose a `<chapter>` contains two `<exercises>`, and you want to have two `<solutions>` within the chapter, each covering just one of the `<exercises>`. This can be accomplished with `@scope`, and you can arrange the four divisions (two `<exercises>` and two `<solutions>`) however you wish within the chapter.

An author filters the types of exercises, and their components, through attributes of the `<solutions>` element. For example

```
reading="hint answer"
```

would cause every `<exercise>` within each `<reading-questions>` to have its `<hint>` and `<answer>` displayed, but not its `<statement>` nor its `<solution>`. These are the attribute names and the possible values.

Table 4.13.1 Attributes (left) and Values (right) for `<solutions>` element

`inline`	`statement`
`divisional`	`hint`
`reading`	`answer`
`worksheet`	`solution`
`project`	

So, PreTeXt source like

```
<section>
  <title>Tropical Bird of Paradise<title>
  ...
```

```
    <solutions worksheet="hint solution" project="hint solution">
      <title>Hints and Solutions to Worksheets and Projects<title>
    </solutions>
  </section>
```

would generate an entire subsection with hints and solutions to every worksheet and every project, located anywhere (including in subsections and subsubsections) in the section on Birds of Paradise.

An `@admit` attribute specifies some feature of an exercise's serial number to determine whether its components are admitted into the solutions division. (For example, the "serial number" of Exercise 1.2.3 is 3.) Presently, the only options are odd, even, and the default `all`. So, PreTeXt source like

```
  <solutions divisional="answer" admit="odd">
```

would generate a subsection with answers to only the odd-numbered divisional exercises.

4.14 Images

4.14.1 Raster Images

A **raster image** is an image described pixel-by-pixel, with different colors and intensities. Photographs are good examples. Common formats are Portable Network Graphics (PNG) and Joint Photographic Experts Group (JPEG, JPG), which will both work with `pdflatex` and modern web browsers. JPEG is a good choice for photographs since they are compressed on the assumption they will be viewed by a human, while PNG is a lossless format and good for line art, diagrams and similar images (if you do not have vector graphics versions, see below).

To use these images, you simply provide the complete filename, with a relative path. A subdirectory such as `images` is a good choice for a place to put them. It is your responsibility to place these images where the LaTeX output will find them or where the HTML output will find them. Your PreTeXt source would look like:

```
<image source="images/crocodiles.png" width="50%"/>
```

Typically you would wrap this in a `<figure>` that might have an `@xml:id` attribute for cross-references, with or without a caption. There is no `@height` attribute, so the aspect ratio of your image is your responsibility outside of PreTeXt. The `@width` attribute is a percentage of the available width of the text (outside of a `<sidebyside>` panel).

You should also provide a `<description>` and/or `<shortdescription>` as a child of the `<image>`, or else explicitly declare the image to be decorative with an attribute `@decorative` set to yes. A `<shortdescription>` should be text-only (but perhaps with `<var>` children) and be less than 100–140 characters long. A `<description>` should be structured with `<p>` and `<tabular>` elements. For example:

```
<image source="images/crocodiles.jpeg" width="50%">
    <shortdescription>Five crocodiles partially submerged.</shortdescription>
    <description>
        <p>
            Five crocodiles are in a pond. Three of them have their eyes above
            the water line, looking in the direction of the camera. The other
            two are in the background and only their tails are visible above
            the water line.
        </p>
    </description>
</image>
```

See Subsection **??** for advice on writing effective image descriptions.

4.14.2 Vector Graphics

An image is a **vector graphic** if the file describes the geometric shapes that constitute the image. So a simple diagram would be a good candidate, but a photograph would not. Popular formats are Portable Document Format (PDF) and

Scalable Vector Graphics (SVG). You will get the best results with PDF images in LaTeX output and SVG images for HTML. The principal advantage of these formats is that they scale (big or small) smoothly, along with fonts. This is critical when you cannot predict the screen size for a reader of an electronic version.

Unless you describe these images with a language (see Subsection 4.14.3), you are responsible for providing the PDF and SVG versions. The pdf2svg utility is very useful if you have PDF images only. To have these different images used for different output formats, you simply follow the instructions above, but *do not include a file extension*. This alerts the conversion to use the best possible choice for any given output, and to embed it correctly. So presuming you made available the files images/toad-life-cycle.pdf and images/toad-life-cycle.svg, the following example would incorporate the PDF version with LaTeX output and the SVG version for HTML output.

```
<image source="images/toad-life-cycle" width="85%">
    <description>The four stages of a toad's life.</description>
</image>
```

Vector graphics images can be created with source code in different languages (Subsection 4.14.3) or with applications, such as Inkscape (Section ??). If you are creating non-technical graphics that have lots of geometric shapes and simple text (a look like a movie poster), then using a tool like Inkscape is a great choice since its native file format is an enhanced version of SVG and a faithful PDF is easy to create.

4.14.3 Images Described by Source Code

There are various languages which may be used to describe diagrams, geometric objects, or data plots. A key strategy enabled by PreTeXt is to put these specifications of such images *directly in your document's source* rather than losing track of them over time.

So we have various elements which are children of <image> that hold these source code descriptions. Then PreTeXt provides techniques for realizing these in the best formats for various devices and print. So if you are accustomed to the idea of a @source attribute pointing to a file, think of these elements as alternative specifications.

4.14.3.1 Asymptote

Asymptote[1] is a vector graphics language that produces high-quality output in WEBGL, SVG, PNG, and PDF formats. You can describe 2-D or 3-D objects, and the 3-D objects are interactive in online output as HTML WEBGL files. LaTeX may be used, and your macros are automatically available for use.

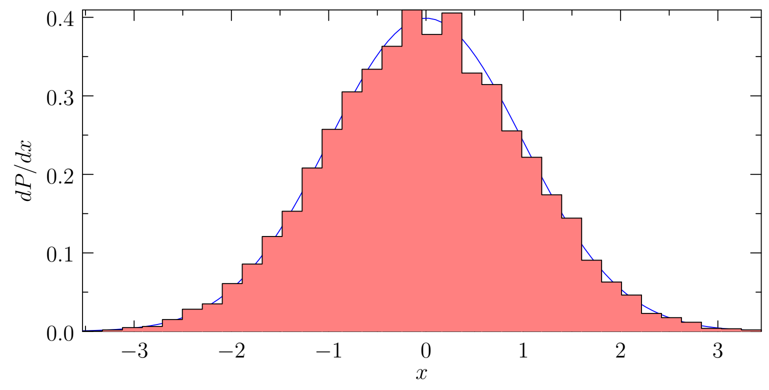
Authoring is straight-forward. Inside an <image> include a child <asymptote> to hold the code. For example:

```
<image xml:id="gaussian-histogram">
    <description>A histogram of Gaussian data.</description>
    <asymptote>
    import graph;
    import stats;

    size(400,200,IgnoreAspect);

    int n=10000;
    real[] a=new real[n];
    for(int i=0; i < n; ++i) a[i]=Gaussrand();

    draw(graph(Gaussian,min(a),max(a)),blue);

    // Optionally calculate "optimal" number
    // of bins a la Shimazaki and Shinomoto.
    int N=bins(a);

    histogram(a,min(a),max(a),N,normalize=true,low=0,lightred,black,bars=false);
```

[1] asymptote.sourceforge.io/

```
    xaxis("$x$",BottomTop,LeftTicks);
    yaxis("$dP/dx$",LeftRight,RightTicks(trailingzero));
    </asymptote>
</image>
```

Here is the result. Look elsewhere for examples of 3-D output from Asymptote.



Notes:

- Notice the necessity of escaping the less-than in the for-loop. See Subsubsection 4.1.4.2.
- Setting a `@xml:id` is necessary to have a stable name for graphics files that will be generated.
- The `<description>` is an important part of making your output accessible.
- Notice the use of LaTeX for the label on the vertical axis. All of your macros defined in `docinfo` are available for use, so you can keep notation consistent.
- You need to produce PDF versions of your diagrams for use in a conversion to LaTeX.
- You need to produce HTML versions of your diagrams for use in a conversion to an electronic format based on HTML. For a 2-D diagram these are a thin wrapper around an SVG image. For a 3-D diagram these are interactive WebGL objects.

  It is very important to note that these HTML versions contain the height and width of the diagram and these are queried by a conversion of your document to HTML format in order to compute the aspect ratio. Therefore they need to be available with your other source files (typically in an `images` directory). So in a very real sense these files become part of your source.
- You may want to produce SVG versions of your diagrams for conversion to EPUB, and PNG versions for conversion to the EPUB precursor for Kindle format.
- PDF versions produced by the `pretext` script will not include the RPC extensions. So “rotatable” 3D images rendered by the proprietary viewer, Adobe Acrobat, are not created, consistent with our open source philosophy.
- Colors in Asymptote can be hard-coded using `rgb` syntax. Colors can also be defined at the top of an Asymptote file, to be referred to later. You may wish to produce PDF in both color (electronic) and black and white (print on demand) formats, and you probably do not want to maintain parallel source for both versions. Rather than writing (for example) `pen p=rgb(0,0,.7);` in your Asymptote code, you can write `pen p=curvepen1;`. Then, in the `<docinfo>` section of your document, you can add an `<asymptote-preamble>` and include the line `pen curvepen1=rgb(0,0,.7);`. Once you are ready to produce your black and white version, you need only change the definition of `curvepen1` in your `<asymptote-preamble>`.

  One note of caution: if your preamble includes Asymptote code that only works once certain libraries are loaded, you must include lines to import those libraries in your preamble. For example, to define a `material` you must first have the line `import three;`.

Asymptote may be run as a program installed locally, but the project also has an on-demand online server written by Supakorn Jamie Rassameemasmuang. By default, the `pretext/pretext` script (Chapter ??) will interface automatically with this server to create your diagrams. Furthermore, Asymptote provides a very useful web application[2] written by Pedram Emami. This is a great place to learn, experiment, and iterate as you become more skilled at building high-quality graphics to illustrate the concepts in your document.

Best Practice 4.14.1 Build 3-D Asymptote Figures. If your project uses geometric or mathematical objects that are three-dimensional, invest some time in learning the Asymptote vector graphics language. The interactive diagrams for your HTML output produced by Asymptote, in WEBGL format, are outstanding and will greatly enhance your project. (And the other static formats are similarly excellent.) The `pretext/pretext` script will create these diagrams, in the necessary formats, with no extra software by using an online server.

4.14.3.2 Images in LaTeX Syntax

There are a variety of LaTeX packages for authoring a diagram, plot, or graph. Examples include: TikZ, PGF, Xy-pic, and PSTricks. Generally, the `<latex-image>` tag allows you to incorporate this code into your source and PreTeXt realizes these descriptions as images in your output.

For LaTeX output the procedure is transparent—PreTeXt simply incorporates the preamble information and the image's code in the correct places in the LaTeX output, scaled to fit whatever space is described on the `<image>` element. Then traditional LaTeX processing will do the right thing. For output to other non-LaTeX formats, such as HTML or EPUB, we need some help from the CLI to generate other formats. This tool will isolate the image's code and bundle it up with the necessary preamble to make a complete single-purpose LaTeX file. Once converted by LaTeX to a PDF version, other tools can convert the image into other formats, such as SVG. In this way, you can use LaTeX packages for describing images, use mathematically-correct labels in LaTeX syntax, and use your own macros for consistency in notation, yet also employ the resulting images in more modern output formats. Note that as of 2020-07-24, limited testing indicates that PSTricks needs to be processed with the `xelatex` engine, and the `pstricks-add` package might also be necessary. Any updates, especially using `pdflatex` would be appreciated. Finally, processing with `xelatex` might be necessary if your labels use Unicode characters.

Much like the `<asymptote>` tag, the `<latex-image>` tag is used as a child of `<image>` and can be thought of as an alternative to the `@source` attribute of `<image>`. The contents need to be a complete specification of the image. For example, a TikZ image will typically begin with `\begin{tikzpicture}`. Inside of your document's `<docinfo>` you will likely need to employ a `<latex-image-preamble>` element to hold necessary `\usepackage` commands and any global settings, such as the style for tick-marks and labels on axes of graphs. The source code in this next example is greatly abbreviated and mildly edited, see the source for the complete example.

```
<docinfo>
    <latex-image-preamble>
    \usepackage{tikz}
    </latex-image-preamble>
</docinfo>

<figure>
    <caption>RNA Codons Table, by Florian Hollandt</caption>
    <image xml:id="rna-codons-table" width="100%">
        <description>A table of the RNA codons.</description>
        <latex-image>
        \begin{tikzpicture}
        \footnotesize
        \tikzstyle{every node}=[inner sep=1.7pt,anchor=center]
        % to_x and from_x styles denote bonds terminating
        % or starting in labeled nodes. x denotes the
        % number of letters in the node label.
        \tikzstyle{to_1}=[shorten >=5pt]
        \tikzstyle{to_1i}=[shorten >=6pt]
```

[2] `asymptote.ualberta.ca/`

```
        \tikzstyle{to_2}=[shorten >=7pt]
        \tikzstyle{to_3}=[shorten >=8pt]
        ...
        \begin{scope}[scale=0.5]    % Asparagine
        \draw[ultra thick,shorten >=2pt,shorten &lt;=2pt] (90-2*5.625:8.2)
                        arc(90-2*5.625:90-4*5.625:8.2);
        \path (90-3.5*3.625-3:13.3) node (zero) {};
        \draw[to_2]  (zero.center)  -- ++(30:1) node (CO) {}
                        -- +(330:1) node [anchor=base] {O$^{\mbox{-}}$};
        \draw[to_1]  (CO.center)    -- +(90:1) node (Od) {O};
        \draw[to_1i] (CO.30)        -- +(90:1);
        \draw[to_3]  (zero.center)  -- ++(150:1) node {NH$_{\mbox{3}}^{\mbox{+}}$};
        \draw[to_2]  (zero.center)  -- ++(270:1) node(Cb){}
                        -- ++(330:1) node (Cc) {}
                        -- +(30:1) node (Cd) {NH$_{\mbox{2}}$};
        \draw[to_1i] (Cc.center)    -- +(270:1) node (O) {};
        \draw[to_1]  (Cc.210)       -- (O.150);
        \path (O.center) node {O};
        \end{scope}
        ...
        \node at (90-55*5.625:4.5) {C};
        \node at (90-58*5.625:4.5) {S};
        \node at (90-61*5.625:4.5) {L};
        \node at (90-63*5.625:4.5) {F};
        \end{tikzpicture}
        </latex-image>
    </image>
</figure>
```

This will result in:

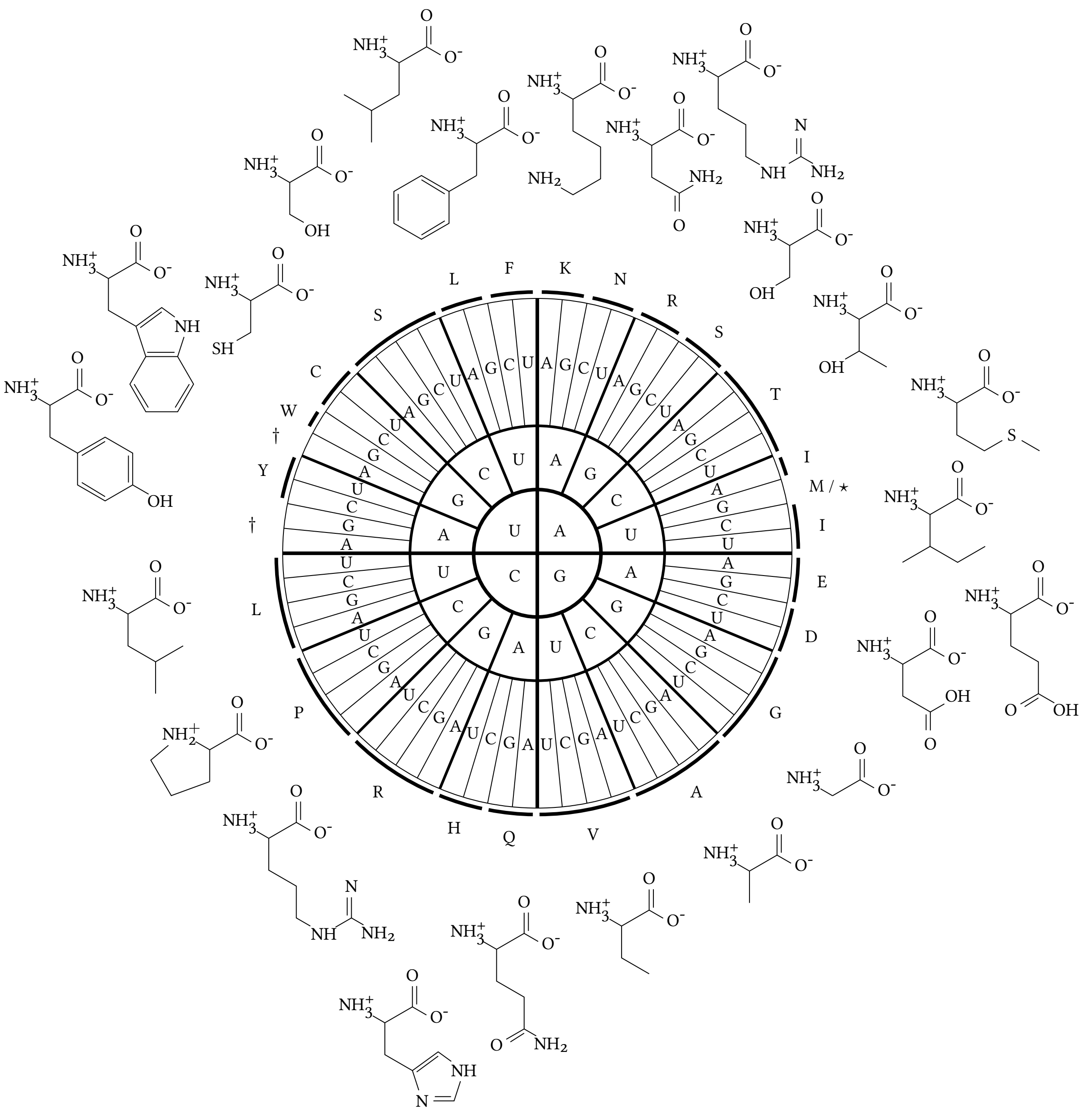


Figure 4.14.2 RNA Codons Table, by Florian Hollandt, from TeXample.net[3]

4.14.3.3 Scaling TikZ Images

Images authored in TikZ[4] are the most popular. Text (nodes) in a TikZ picture are at whatever the current LaTeX font size is. The other parts of the picture (lines, circles, rays, etc.; the “line art”) can be scaled as part of an overall scale factor. The point being, the scale factor will not scale the text simultaneously. It is not unlike a map application on your mobile device. The name of a road is too small to read, so you zoom in on the map, making the street bigger, but the name stays in the same font size and is still unreadable. This means some manual labor is involved when you place a TikZ picture into a PreTeXt document.

For many authors, the goal is to have the text in their TikZ picture have the same size as the surrounding text,

[3] `texample.net/tikz/examples/rna-codons-table/`
[4] `github.com/pgf-tikz/pgf`

both in a PDF and in HTML. We now explain how to accomplish this consistently.

Preparation. Well *before* designing many TikZ images, answer the following questions.

1. For your LaTeX output, what will the overall font size be?
2. For your LaTeX output, what will the width of the text block be? Note that this will normally be computed by PreTeXt, dependent on your chosen font size. A larger font will mean a greater width. You can generate the LaTeX source file and look early in the preamble to see what width is being used. It is also possible that you may be setting this with yourself (Section **??**). The ratio of line width to font size is always $34:1$.

TikZ in LaTeX. Every image in PreTeXt may be constrained by width and/or margins, or may be restricted to a panel of a `<sidebyside>` with a certain width. So the TikZ code you author will create an image that is then scaled by PreTeXt to fit the constraints (much as any other image is scaled). Except this is done in a way that scales both the font and the line art. Your main goal is to have this scaling use a scale factor of 1.0. Which, of course, sounds like a waste of effort, but it is critical for how the image behaves in HTML (next).

To accomplish this unit scaling, follow this procedure for each TikZ picture.

1. Determine the width of the TikZ picture itself, in physical units of length. Typically, the lengths used for larger portions are described in centimeters. But note that an overall scale factor is sometimes applied for convenience (or as a result of poor planning!). Also, the default unit length (centimeters) can be changed. Note also, that text may "push out" to the right and left, defining the boundaries on the sides, and these lengths can be hard to compute or predict.
2. Recall the width of your text (above). Recognize that list items will be indented (reducing width), and perhaps there are multiple indents if a list has multiple levels.
3. Now you want the width of your picture as a percentage of the overall available width. By default, your overall width will be points, and your picture width will be in centimeters. You may be familiar with a "big point" (or "desktop publishing point") which is 72 points to the inch. TeX however uses 72.27 points to the inch, which makes a TeX point equal to 0.03514598 centimeters. Convert to whatever common unit makes sense to you, since it is the dimension-less ratio you are after.
4. Use this percentage as the `@width` attribute on the `<image>` (with a percent sign).

Now produce a PDF and you will find that the font in the surrounding text, and the font in image, will match identically. I like to check this carefully by zomming in on the PDF and using an on-screen pixel ruler to check the heights of identical letters. KRuler[5] is one such example for Linux, suggestions for other operating systems are welcome.

Note that in practice you will envision your picture as large or small, and you will begin with some overall physical width in mind, relative to the line width.

TikZ in HTML. For HTML output, the goal is to not edit your source. In other words to not change the `@width` attributes that have been so carefully computed and to not edit the TikZ code. But you will want to maintain fidelity with the surrounding font.

HTML output is designed to behave very similarly (not identically) to how LaTeX output behaves. In other words, the ratio of line width to font size is $34:1$. In this way, line length and font size are such that a long paragraph will usually have an identical number of lines in LaTeX (at any font size) and in HTML.

Our tools produce Scalable Vector Graphics (SVG) versions of TikZ pictures for use in HTML output. Being scalable means a reader can zoom in without any pixelation of the images. This is helpful for those with low vision, or if some fine point of a picture needs to be examined closely. It also means an SVG can be scaled by any factor when placed in PreTeXt HTML. However, the work done for a unit scaling for LaTeX output will continue to provide the correct scaling for HTML! (Provided the text width used for the PDF production is the one automatically computed from the font size via the $34:1$ ratio.)

[5] `apps.kde.org/kruler/`

Example 4.14.3 Case Study: Scaling a TikZ picture. The PreTeXt source below describes a simple TikZ picture, nd is followed by the picture itself. The rectangle is 8 centimeters wide. The Guide is produced as a PDF with 10 point text and a text width of 6.5 inches. (This is too wide for comfortable reading, and contrary to our recommendations. See Best Practice **??**.) Normally, a choice of 10 point text would result in a width of 340 point, or about 4.7 inches.

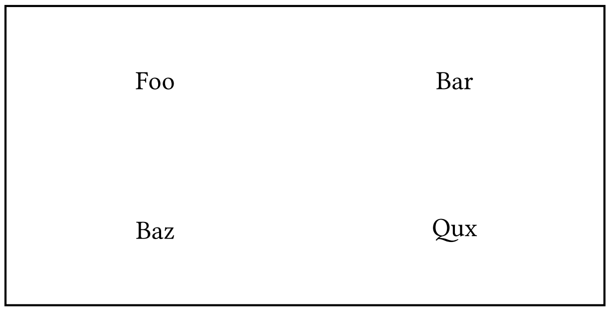
So we compute the fraction of the available width required, as a percentage:

$$\frac{8\text{ cm}}{6.5\text{ in}} = \frac{8\text{ cm}\left(\frac{1\text{ in}}{2.54\text{ cm}}\right)}{6.5\text{ in}} = 0.4846 = 48.46\%$$

and we use that as the width of the image.

```
<image xml:id="scaling-tikz" width="48.46%">
    <latex-image>
    \begin{tikzpicture}
    % 1 cm is default unit of length
    % a rectangle: 8 cm wide, 6 cm tall
    \draw[draw=black, thick] (4,2) rectangle (-4,-2);

    \node at (-2,  1) {Foo};
    \node at ( 2,  1) {Bar};
    \node at (-2, -1) {Baz};
    \node at ( 2, -1) {Qux};
    \end{tikzpicture}
    </latex-image>
</image>
```

Some characters for comparison: Foo Bar Baz Qux



In the PDF version, the text matches between the image and the surrounding text almost identically. We could slide the image right and left by adjusting the margins (the default is to be centered). But if we want the image bigger and smaller, we need to adjust the TikZ code and recompute the `@width` attribute.

Now for HTML we need to produce an SVG version that is a close match. The HTML version is a close match for LaTeX built with a computed text width (for any font size). We do not want to change the percentage of the width devoted to the TikZ picture, and we do not want to change the TikZ code itself. If we had not chosen a different text width (the 6.5 inches, versus a computed 340 point), then we could generate the SVG by supplying the same publication file, so as to use the same font size. However, our text width is 38% larger in the LaTeX version,

$$\frac{6.5\text{ in}}{340\text{ pt}} = \frac{6.5\text{ in}\left(\frac{72.27\text{ pt}}{1\text{ in}}\right)}{340\text{ pt}} = 1.3816$$

The font size needs to increase by a similar percentage,

$$10\text{ pt} \times 1.3816 = 13.816\text{ pt} \approx 14\text{ pt}$$

So we generate the SVG image with a *different* publication file, giving a font size of 14 point. The HTML font in the text may be very different from the LaTeX font used in the TikZ picture, but their *sizes* are nearly identical. Note that our use of LaTeX only supports 8 different font sizes, so it was fortuitous in this example that the 38% increase was so close to the supported 14 point font size. Note also, that since we used a different text width for the PDF, the resulting 40% increase in the font size for the SVG could play havoc with text that has been placed carefully not to overlap other components of the picture. □

There are myriad ways to scale and transform a TikZ picture. You might choose to intentionally use a smaller font size than the surrounding text, as in Figure 4.14.2. Or, fidelity with the surrounding text might not be important to you. Or you might prefer that images perform better in HTML. But hopefully the above discussion and example provide enough insight into how the various constructions behave. The important points are:

- TikZ uses physical units for the overall width of a picture, and nodes have text using the ambient font size of the PreTeXt LaTeX file (unless prescribed otherwise).
- PreTeXt scales a TikZ picture uniformly (text *and* line art) to fit into constraints given in the source.
- The SVG version of a TikZ picture is also uniformly scalable and at the same width as the original will have text of the correct font size. However, when used in HTML output, it is scaled on the assumption that the ratio of the line width to the font size is 34:1. This is the default width computed by PreTeXt for all supported font sizes. Changes in this ratio for PDF production requires an equivalent change in font size during SVG construction, via the publication file.

4.14.3.4 Images in Sage Syntax

Sometimes the necessary computations for an image are not part of the capabilities of whatever system is actually realizing the image. We have good support for Sage in other parts of your document, and Sage has an extremely wide variety of computational capabilities, in addition to letting you program your own computations in Python syntax with the full support of the Sage library. Rather than translating Sage output as input to some other graphics program, we simply capture the graphics output from Sage. So if your graphics are derived from non-standard, or intensive, computation this might be your best avenue.

Use the `<sageplot>` element, in a manner entirely similar to the `<asymptote>` element and the `<latex-image>` element, as a child of `<image>`, and containing the necessary Sage code to construct the image. There is one very important twist. The last line of your Sage code ***must*** return a Sage `Graphics` object. The `pretext/pretext` script (Chapter **??**) and PreTeXt-CLI (Section **??**) will isolate this last line, use it as the right-hand side of an assignment statement, and the Sage `.save()` method will automatically be called to generate the image in a file. Note that there are four different file types, depending on if the graphic is 2D or 3D, and the output format of the conversion.

The `@variant` attribute of the `<sageplot>` element may be `2d` or `3d`, since PreTeXt is not capable of analyzing your Sage code. The default value is `3d` so can be skipped for 2D plots. For technical reasons, it is also necessary to specify the aspect ratio of a graphic for the 3D case using the `@aspect` attribute. The value can be a positive real number (decimal) or a ratio of two positive integers separated by a colon. The default is a square (`1.0` or `1:1`).

Table 4.14.4 File formats for `sageplot` images

	2D	3D
LaTeX	PDF	PNG
HTML	SVG	HTML (for `iframe`)

Note that the PNG images in the 3D case are not very good. This needs help on the Sage side. And since 3D images in HTML output are inserted via an HTML `iframe` they can misbehave if you do not get the aspect ratio close to right. On the plus side, the 3D HTML images may be manipulated interactively with keyboard arrow keys, a mouse scroll wheel, and by dragging with a mouse using both a left and a right mouse press.

Pay very careful attention to the requirement that the last line of your code evaluates to be a graphics object. In particular, while `show()` might appear to do the right thing during testing, it evaluates to Python's `None` object and that is just what you will get for your image. The example below illustrates creating two graphics objects and combining them into an expression on the last line that evaluates to the graphics object that will be created in the desired graphics files.

```
<figure>
    <caption>Negative multiple of a curve</caption>
    <image xml:id="negative-curve" width="65%">
```

```
        <description>Plot of x^4 - 1 and its negative.</description>
        <sageplot>
        f(x) = x^4 - 1
        g(x) = -x^4 + 1
        up = plot(f, (x, -1.5, 1.5), color='blue', thickness=2)
        down = plot(g, (x, -1.5, 1.5), color='red', thickness=2)
        up + down
        </sageplot>
    </image>
</figure>
```

This will result in:

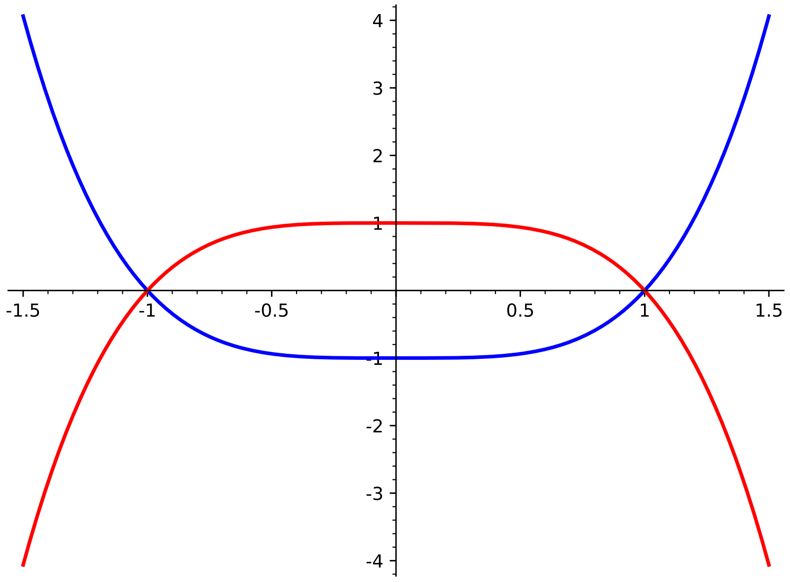


Figure 4.14.5 Negative multiple of a curve

Note the necessity of using the `pretext` script (Chapter ??) to independently invoke Sage, no matter what sort of output is being created for your document.

4.14.4 Image Formats

Best Practice 4.14.6 Preferred Image Formats. The best formats for images, in order, are:

SVG	Vector graphics format ideal for HTML output formats. Scalable and compact. Converts to other formats, such as PDF.
PDF	Vector graphics format ideal for print and PDF output formats. Good tools exist to convert back-and-forth between SVG and PDF.
PNG	Lossless and compressible format for raster images. May be used for both HTML and PDF outputs.
JPEG	Compressed lossy format which works well for photographs. May be used for both HTML and PDF outputs. PNG should be preferred when there is a choice, except in the case of a photograph. Converting between these formats is unlikely to be an improvement.

4.14.5 Image Archives

As an instructor, you might want to recycle images from a text for a classroom presentation, a project handout, or an examination question. As an author, you can elect to make images files available through links in the HTML version, and it is easy and flexible to produce those links automatically.

First, it is your responsibility to manufacture the files. For making different formats, the `pretext` script can sometimes help (Chapter ??). The Image Magick `convert` command is a quick way to make raster images in different formats, while the `pdf2svg` executable is good for converting vector graphics PDFs into SVGs. Also, to make this easy to specify, different versions of the same image must have identical paths and names, other than the suffixes. Finally, the case and spelling of the suffix in your PreTeXt source must match the filename (e.g. `jpg` versus `JPEG`). OK, those are the ground rules.

For links for a single image, add the `@archive` attribute to the `<image>` element, such as

```
<image  ...  archive="pdf svg">
```

to get two links for a single image.

To have every single image receive an identical collection of links, in `docinfo/images` place an `<archive>` element whose content is the space-separated list of suffixes/formats.

```
<archive>png JPEG tex ods</archive>
```

will provide four links on every image, including a link to an OpenDocument spreadsheet.

For a collection of images that is contained within some portion of your document, you can place an `@xml:id` on the enclosing element and then in `docinfo/images` place

```
<archive from="the-xml-id-on-the-portion">svg png</archive>
```

to get two links on every image *only* in that portion (chapter, subsection, side-by-side, etc.). The `@from` attribute is meant to suggest the root of a subtree of your hierarchical document. If you use this, then *do not* use the global form that does not have `@from`.

You may accumulate several of the above semi-global semi-local forms in succession. An image will receive links according to the last `<archive>` whose `@from` subtree contains the image. So the strategy is to place general, large subtree, specifications early, and use refined, smaller subtree specifications later. For example,

```
<archive from="the-xml:id-on-a-chapter">svg png</archive>
<archive from="the-xml:id-on-the-introduction">jpeg</archive>
<archive from="the-xml:id-on-a-section-within" />
```

will put two links on every image of a chapter, but just one link on images in the introduction, and no links at all on every image image within one specific section. Again, do not mix with the global form. You can use the root document node (e.g. `<book>`) for `@from` to obtain a global treatment, but it is unnecessary (and inefficient) to provide empty content for the root node as first in the list—the same effect is the default behavior.

Notice that this facility does not restrict you to providing files of the same image, or even images at all. You could choose to make data files available for each data plot you provide, as spreadsheets, or text files, or whatever you have, or whatever you think your readers need.

Finally, “archive” may be a bit of a misnomer, since there is no historical aspect to any of this. Maybe “repository” would be more accurate. Though for a history textbook, it might be a perfect name.

4.14.6 Copies of Images

Sometimes you want to use the same image more than once. Putting it in a `<figure>` and then making a cross-reference (`<xref>`) can work well in HTML output since it will be available as a knowl. However in a static format (PDF, print) the reader will need to chase the cross-reference.

For a raster image, you can just point to the file again with the `@source` attribute. You are free to wrap it in a `figure` and thus change the caption. It will get a new number as a new figure, and you will need to assign a new unique `@xml:id` attribute. Maybe appending `-copy-2`, or similar, to the `@xml:id` will be helpful.

If you have a figure generated from source code (such as in TikZ) you really do not want to edit and maintain two copies that may not stay in sync. Instead, you can place the code into a file and `xinclude` it twice. Study Section ?? carefully, and note that this is an excellent place to take advantage of setting the `@parse` attribute to `text` (no need to escape problematic XML characters). Notice that when you generate images, you will have two image files with identical contents, but different names. This is no place for false optimization. Bits are cheap and your time is valuable. It is far more important to only maintain a single copy of the source, than to be caught up with the "waste" of having two copies of the same file (and which are managed for you). We demonstrate this with the sample book, since it is all set up with the `xinclude` mechanism. See the two plots of the 8-th roots of unity in the complex numbers section of the chapter on cyclic groups.

4.15 Programs and Consoles

4.15.1 Content and Placement of Programs and Consoles

A `<program>` will be treated as verbatim text (see Section 3.16), subject to all the exceptions for exceptional characters (see Section 3.14). Indentation will be preserved, though an equal amount of leading whitespace will be stripped from every line, so as to keep the code shifted left as far as possible. So you can indent your code consistently along with your XML indentation. For this reason it is best to indent with spaces, and not tabs. A mix will almost surely end badly, and in some programming languages tabs are discouraged (e.g. Python).

A `<console>` is a transcript of an interactive session in a terminal or console at a command-line. It is a sequence of the following elements, in this order, possibly repeated many times as a group: `<input>`, and `<output>`. The `<output>` is optional. The content of these two elements is treated as verbatim text (see Section 3.16), subject to all the exceptions for exceptional characters (see Section 3.14). A `@prompt` attribute on the `<input>` can be supplied to provide a system prompt distinct from the actual commands. The default prompt is a dollar sign followed by a space. If it is more convenient `@prompt` may be supplied on the `<console>`, to be used in each enclosed `<input>`. If you do not want any prompts at all, just use an empty value for `@prompt`.

A `<program>` or `<console>` may be wrapped in a `<listing>`, see Section 4.19. This will behave similar to a `<figure>`, with the `<caption>` displayed below, and a number assigned. So, in particular, if your program or console is important enough to cross-reference, it is an enclosing `<listing>` that serves as the target.

The `@language` attribute may be used to get some degree of language-specific syntax highlighting and/or interactive behavior. We will eventually provide a table of attribute values here. They are always lowercase, and a good first guess is likely to succeed.

In some output formats, notably HTML, the syntax highlighter can add lines or highlight individual lines of code. Highlighting lines is not supported for LaTeX output. To display line numbers, set the `@line-numbers` attribute to `yes`. To highlight particular lines, set `@highlight-lines` to a comma-separated list consisting of individual lines and/or ranges indicated with dashes. Some examples are: `5`, `2,5`, `2,5-8,10-15,15`.

For interactive versions hosted on Runestone servers, the `@label` attribute is critical, just like for interactive exercises. So be certain to read Best Practice 4.12.1.

4.15.2 Interactive Programs, CodeLens

CodeLens is an interactive version of a computer program, which can be visualized by stepping through the code one statement at a time, watching output, variables, and other data structures change. So it is similar to a debugger, except the reader does not set breakpoints or modify program data on-the-fly. This is possible automatically for several different languages when your HTML is hosted on a Runestone server (Chapter **??**). This may also be accomplished "in browser" when hosted on any old generic web server. The catch is that for a generic server a publisher must generate **trace data** in advance, typically with the PreTeXt-CLI (Section **??**). Place the `<interactive>` attribute on a `<program>` element with the value `codelens` to elect this behavior (`no` is the default value). Also, be sure to specify a language from the supported languages: Python, Java, C, and C++. Consult Table 4.15.1 below for a summary of various combinations. When an output format does not support an interactive CodeLens instance, the fallback is a static program listing.

4.15.3 Interactive Programs, ActiveCode

ActiveCode is an interactive environment where a reader may work on code through repeated edit-compile-test cycles. Code can be provided by an author as a complete program to be modified, a partial program to be completed, or nothing at all. One good example is that maybe header files, import statement, and similar are provided, and a skeleton of a main entry-point procedure is also provided. Then a reader can concentrate on the more conceptual parts of the programming. Some languages will be executable "in browser" on any old generic web server, while others must be on a Runestone server (Chapter ??) where a Jobe Server[1] is running to support the execution.

Place the `@interactive` attribute on a `<program>` element with the value `activecode` to elect this behavior (`no` is the default value). Also, be sure to specify a language from the supported languages. Consult Table 4.15.1 below for a summary of various combinations. When an output format does not support an interactive ActiveCode instance, the fallback is a static program listing.

For languages that support CodeLens, a button will be created that allows the reader to step through the program. For some programs, especially ones using libraries like turtle graphics or image, CodeLens will not function. To prevent showing the CodeLens button on programs for it will not work, authors can set the `@codelens` attribute to `no`.

Note also that a data file may be provided independently for consumption by an ActiveCode program. See Section 4.16.

ActiveCode elements that are language `sql` can make use of an SQLite database file. This is different than a `datafile` (which is expected to be text or an image). To include a database file, use the `@database` on the `<program>` element and specify the file to load as a string relative to the `@@external` top-level directory

If you want to include code from one or more preceding `<program>` elements, use the `@include` attribute whose value is a list (comma-separated or space-separated) of `@xml:ids` for the code you want included.

4.15.4 Interactive Program Capabilities

This table lists which types of interactivity are available on various combinations of servers and programming languages. The entry "AC + CL" means that both ActiveCode and CodeLens instances are availble, but the ActiveCode instance will have a CodeLens button enabled.

Note that `python` is generic Python with the standard libraries (version 3.x). On the other hand `python3` is a Runestone server installation (only) with a number of additional popular Python packages available, such as `numpy` and `pandas`.

Table 4.15.1 Interactive Programs

Language	**`@language`**	**Server**	
		Generic	**Runestone**
Python	`python`	AC + CL	AC + CL
Python 3	`python3`		AC
Java	`java`	CL	AC + CL
C	`c`	CL	AC + CL
C++	`cpp`	CL	AC + CL
JavaScript	`javascript`	AC	AC
HTML	`html`	AC	AC
SQL	`sql`	AC	AC
Octave	`octave`		AC

AC = ActiveCode, CL = CodeLens

4.16 Data Files

In concert with interactive programs (see Section 4.15) you can define a file of data that may be employed by those programs. The necessary element is `<datafile>`. It requires a `@label` attribute. The `@filename` is also required and is the name the file is known by in an ActiveCode program. Do not try to impose any sort of directory structure on this name. Just a filename. In the case of a text file (see below), the `@editable` attribute is optional. The value `no` is the default, with `yes` as the other option. The attributes `@rows` and `@cols` are optional for text files, and default to `20` and `60` respectively. Finally, a non-editable text file (only) may have its contents hidden by setting the `@hide` attribute to `yes`, rather than the default value of `no`.

[1] `github.com/trampgeek/jobe`

Where might you place a <datafile>? Lots of places are possible, such as in an <example> or a programming <exercise>, close to an ActiveCode <program>. So, in expository material or in activities for readers to work through. The purpose-built COMPUTATION-LIKE block, <data>, which will get you a heading, number, title, cross-reference target, etc. (see List 4.2.2), is an option if the file itself needs more prominence or dedicated explanation. Notice that this feature is very powerful, and thus requires a bit of machinery to support. If you just want to point your reader to a file (and leave them to work with it outside of your project), either globally or locally, the read about the <dataurl> element at Section 4.16.

Text as Data Files. Inside of a <datafile> place an <pre> element. There are then two options: provide the contents of the text file right in your source PreTeXt file, as you might for other preformatted text, or supply a @source attribute whose value is the name of an external text file you provide. The former is appropriate for "toy" examples, while the latter may be used for "serious" files with many lines, or with long lines. Note that if you provide the file as the content of the <pre> element, it can be indented to match your source file indentation, and will undergo some manipulation, such as removing leading whitespace, and ensuring a final newline, but preserving any *relative* indentation. If provided via a @source attribute, there is no manipulation.

Such a text file may be declared editable by the reader, presumably to allow them to witness the resulting behavior of a some employing program. The @rows and @cols attributes describe the viewport into the file provided in the HTML output. Typically scrollbars will allow the reader to survey all of a large file. In static outputs, the first few lines are shown, given by the value of @rows, and lines are truncated according to the value of @cols.

Images as Data Files. Inside of a <datafile> place an <image> element with a @source attribute. As usual, this attribute should be the name of an external file you provide. Most common formats are supported, but it is important to use standard extensions, so the format can be discerned. Now this file may be explored programmatically by opening the file using the name provided in @filename.

Keep the size of the image small, say 300–400 pixels in each direction. You may also supply the usual layout controls, such as @width, and these will be consulted in the formation of output formats. Ideally, you should use a width that scales the image to look something like its "native" resolution, since part of an image-processing exercise may depend on this aspect of the input. HTML output uses a 600 pixel overall width, so a percentage can be computed based on this parameter.

Notes on Data Files. Some notes that apply to each type of data file.

- Note that the name of the data file in a @source attribute need not have any resemblance to the new name given to the file via the @filename attribute. In other words, the reader will never know (or care) what @source was.

- Whenever the @source attribute is used, there needs to be an advance step performed by the CLI Section **??** or the pretext/pretext script (Chapter **??**) to generate an auxillary file (yes, a third file!) to aid the transistion from an external file to a file that can be used by the reader in programs.

- In all cases, for an HTML build the contents of the data file live within an HTML page, as text for a text file, and as a base-64 encoding for an image file. Hence for a non-Runestone build, any employing program *must* be on the same page, and an author should think ahead about the granularity of how a project might be chunked into pages (Section **??**).

  In a build for use on Runestone Academy, the file will be in the Runestone database and usuable throughout.

- Some <program> run entirely in your browser, on Runestone Academy or not. An example is when @language is set to python. Other <program>, such as java will only execute when hosted on Runestone Academy.

  In the latter case, you need to do just a bit more than try to open a data file in your program's code. Include a @datafile attribute on the <program> element that is a list (separated by commas or space) of the filenames for files that will be used (these are the names given by the @filename attribute of the <datafile> element).

4.17 Figures

A <figure> is the most generic and flexible container for planar content. But be sure to read Section 3.13 so you are aware of the other possibilities. A figure has a <caption>, which will typically render below the content (even if authored early as metadata) and serves to provide an extra description of the content. So it may be several sentences long. There is also a <title>, which is typically not rendered as part of the figure. Instead it is used for cross-references, or in a list of figures, to identify the figure. So it should be very short and might just be a phrase, such as "Life Cycle of a Salamander."

An <image> is likely the most frequent content in a <figure>. But you may also place a <video>, <audio>, <sidebyside>, or <sbsgroup>. Once completely implemented, an <interactive> is another possibility. (See Section 4.23 for more about the side-by-side construction.)

A special situation is when a <figure> is a panel of a <sidebyside>, which is itself inside a <figure>. Then the interior figure is **subnumbered**. For example, the exterior figure might be Figure 4.12, and if a panel of the <sidebyside> is the second interior figure it will be Figure 4.12(b). For example,

```
<figure>
    <caption>Salamanders at different life stages</caption>
    <sidebyside>
        <figure>
            <caption>Hatchling</caption>
            <image source="salamader-hatchling.jpg"/>
        </figure>
        <figure>
            <caption>Juvenile</caption>
            <image source="salamader-juvenile.jpg"/>
        </figure>
        <figure>
            <caption>Adult</caption>
            <image source="salamader-adult.jpg"/>
        </figure>
    </sidebyside>
</figure>
```

could result in the entire figure being Figure 4.12 and then the juvenile salamander photograph would be inside of Figure 4.12(b).

4.18 Tables and Tabulars

A <table> is a container that houses a <tabular>, which is the actual rows and columns of table entries.

Note that <tabular> may be constructed using the LaTeX Complex Table Editor[1] tool online and then exported in PreTeXt syntax. This produces verbose PreTeXt syntax that is usually equivalent to much simpler PreTeXt syntax once you understand the borders and alignment considerations below.

4.18.1 Tables

A <table> is similar to other blocks in PreTeXt (Section 4.2) and is most similar to a <figure>. It will earn a number, which is likely to be a part of the text of a cross-reference pointing to the table. Rather than a <caption>, it will have a <title>. The main difference is that the principal content *must* be a <tabular>. Only.

4.18.2 Tabular

A <tabular> is the actual headers, rows, and columns of a table. As discussed above, a typical use is to place it inside a <table>, though it can be placed all by itself, typically in among a run of paragraphs.

[1] www.latex-tables.com

Fundamentally a `<tabular>` is a sequence of `<row>` and each `<row>` is a sequence of `<cell>`, which could also be called "table entries."

4.18.3 Table Cells

A given cell can span multiple columns, by providing the `@colspan` attribute with a value that is a positive number, the cell will extend to occupy additional columns.

4.18.4 Table Rows

A `<row>` of a table is a sequence of `<cell>` elements. Each row should occupy the same number of cells, when considering the `@colspan`, as discussed above.

To achieve *column* headers, you indicate that a *row* contains headers. Typically, the contents of every cell in this row will then be rendered in bold, or some other style. The `<row>` element accepts a `@header` attribute with possible values of `no` (the default), `yes`, or `vertical`. The latter is useful if space is at a premium (which always seems to be the case with tables), and the cells of a column are narrow and the header is long. Note that only the first (top) rows can be treated as column headers and these rows must be contiguous. If you think you need column headers mid-tabular, maybe you really have two tables?

4.18.5 Table Columns

Prior to all of the `<row>` within a `<tabular>`, there may be a sequence of empty `<col/>` elements. Having these is optional, but once there is one, then there needs to be as many as the number of columns of the table. These elements do not have any content that appears in the table, but are used to hold attributes that influence the borders or alignment of the cells within a column. These are described below.

So it should now be clear that, after much consideration, that we have chosen a "row first" approach to describing a table.

To encourage good style, we only support row headers as the first column. So this is a property of the entire `<tabular>`. So the attribute `@row-headers` on `<tabular>` can have values `no` and `yes`, with the former as default. Note that "major" and "minor" row headers should be accomplished in the first column by using indentation for the minor headers. Please make a feature request if you would find this useful.

4.18.6 Table Borders and Rules

You can view each cell of your table as having four borders. Or you can imagine rows and columns separated by horizontal or vertical rules. These additions to your table do not change the arrangement of information into rows and columns (a doubly-indexed data set), though you may think it makes the presentation clearer. But less is actually more.

Best Practice 4.18.1 Vertical Rules in Tables. One of the goals of PreTeXt is to gently guide authors towards good choices in the design of their documents, even if we do not claim to know it all ourselves. Take a close look at Table 4.1.3. What's missing? No vertical rules. Try living without them, you will not really miss them. If you think you need to divide a table into two halves, maybe you really need two tables (and then see the "side-by-side" capabilities, Section 4.23).

In the documentation for his excellent LaTeX package, booktabs[2], Simon Fear gives two rules for what he calls "formal tables": (1) Never, ever use vertical rules, and (2) Never use double rules. We have resisted the temptation to enforce the former and have provided an alternative to the second (three thicknesses). He refers to using tables for layout as creating "tableau." If you are finicky about the look of your work, the first three pages of the documentation is recommended reading.

A given `<cell>` can have a border on its bottom edge, and on its right edge. This is accomplished with the `@bottom` and `@right` attributes. The possible values are `minor`, `medium`, and `major`, which control thickness. (Not every conversion can produce three distinct thicknesses, so this should be considered a hint to the conversion.) A value of `none` is the default behavior when the attribute is not used, but can be given explicitly.

[2] `www.ctan.org/pkg/booktabs`

How to get a left border on the first cell of a row? The `<row>` element allows a `@left` attribute which will put a border on the left end of the row, which is also the left border of the first cell.

How to get a top border on a cell? Put a bottom border on the cell above it. But what if the cell is already in the top row and has no cell above it? The relevant `<col>` element allows a `@top` attribute which will place the necessary border on the top-row cell.

Borders and rules verge on presentation, so we are not concerned about which cell a border (or rule) belongs to. So, generally `@bottom` and `@right` can be used in many places, and the exceptional `@top` and `@bottom` maybe used to get the missing border $n + 1$ for a vertical or horizontal sequence of n cells.

The attributes described for cells may also be used on `<row>`, `<col>`, and `<tabular>`. For example a thick horizontal rule after two rows of headers could be accomplished with

```
<row header="yes">...</row>
<row header="yes" bottom="major">...</row>
```

We will not detail all the combinations that are possible, so experiment and you should be able to create any rational look (and some irrational ones).

4.18.7 Table Cell Alignment

The horizontal alignment of the contents of a `<cell>` can be influenced by the `@halign` attribute with values `left`, `right`, `center`, and for “paragraph cells,” `justify`. Similarly the `@valign` attribute will influence the vertical alignment through values `top`, `middle`, and `bottom`. Default alignments are `left` and `middle`.

To align the cells of an entire `<row>`, `<col>`, or `<tabular>` identically, place the relevant attribute on the relevant element. Note that these choices can be overridden by different values on individual consituents.

4.18.8 Multi-line Cells

A cell of a table may contain more text that fits onto one line. If you know exactly where you want the line-breaks to be, then structure the entire cell as a sequence of `<line>` elements.

Or, if you want the contents of a cell to look and feel more like a paragraph (or several), structure the cell as a sequence of `<p>`, which can contain the usual content of a `<p>`, excepting “larger” content such as display mathematics or lists. Now, in this case, you must constrain the width of the cell’s column, to force the line-breaking necessary to render a paragraph as several lines. Use the relevant `<col/>` element, and specify a percentage of the tabular’s overall width, like this:

```
<col width="40%"/>
```

A paragraph cell can be right-justified with the `@halign` attribute set to `justify`. But be aware that if the column is skinny, this can lead to awkward inter-word spaces.

4.18.9 Breakable Tabulars

A `<tabular>` may be specified as **breakable**, inside of a `<table>` or not. Use the attribute `@break` set to `yes`. (The default is `no`.) This only affects conversions to formats with page breaks, such as PDF. Usually the motivation will be a `<table>` or `<tabular>` that is too long for a page, but even a shorter table can be allowed to page break.

As of 2022-07-28 this is effective for simple tables, but introduces some variations for more complex constructions. This is implemented with the LaTeX `longtable` package, which suggests it may take up to four passes with LaTeX to obtain the final version. It is also not effective for a `<tabular>` that is a side-by-side panel. Consult the sample article for examples where more progress is necessary.

4.18.10 Table Philosophy

The *Chicago Manual of Style* [1, 13.1] says:

> A table offers an excellent means of presenting a large number of individual similar facts so that they are easy to scan and compare. A simple table can give information that would require several paragraphs

to present textually, and it can do so more clearly. ... A table should be as simple as the material allows and understandable on its own; even a reader unfamiliar with the material presented should be able to make general sense of a table.

If you review the twenty tables presented in Chapter 13 of CMOS, that are of the type we implement, you will notice several things.

- Only the first column is ever used for row headings.
- Cells do not span multiple rows. (There is no analogue for `@colspan`.)
- Column headings appear at the top, other than **cut-in heads**, which have a very particular form. (We have not implemented these, but would entertain a feature request.)

While our implementation allows for some presentational elements (borders, rules, alignment) our conversions will presume your table hews to the purposes described by CMOS. In particular, it is not a device for spatial layout of complex elements. You might find that the `<sidebyside>` and `<sbsgroup>` layout devices will suit that purpose better (see Section 4.23).

Best Practice 4.18.2 Tables are Difficult. Width is always at a premium, and then when a `<tabular>` has more than a few columns, the width becomes even more dear. When a `<cell>` has text that looks like a phrase or a sentence, rather than numerical data or symbols, it can be even harder to pack it all in. A common example is a schedule of talks at a small professional conference where each time slot (rows) might have two or three talks simultaneously in parallel sessions (columns).

We offer **paragraph cells** which automatically break lines, but you need to specify a `@width` on the `<col>` as a percentage to indicate where line-breaking happens. For manual line-breaking, a `<cell>` can be structured entirely by `<line>` elements.

The next complication is that the LaTeX used for PDF output tends to make columns as wide as necessary and will not break lines without the devices mentioned in the previous paragraph. The HTML output can sometimes be a bit more forgiving and flexible. So we suggest building the LaTeX output first and getting that right, and then the HTML is likely to follow along and not need much futher refinement.

In contrast to most of PreTeXt, you may need to experiment, refine your approach, iterate, and maybe do things contrary to usual best practices elsewhere. For example, the clickables for URLs and knowls might need to be short and less-informative in order to save some width. Abbreviations, initialisms, and acronyms can also save some width.

4.18.11 Summary: Table Reference

Finally, we summarize the available options for a table with...a table. Because it would take too much text to describe fully.

This table describes how to construct tables via the `tabular` element. The `table` element may be used to enclose the raw table, so as to associate a title and get vertical separation with horizontal centering.

The `tabular` element contains a sequence of `row` elements, and must contain at least one. Each `row` contains a sequence of `cell` elements and must have the same number in each row (acccounting for the use of the `colspan` attribute). The contents of the `cell` elements are the text to appear in entries of the table.

A sequence of `col` elements may optionally be used. But if one appears, then there must be the right number for the width of the table. They are empty elements always, and just carry information about their respective column.

Where the body of the table below has an entry, it means the attribute may be used on the element, and affects the range of the tabular described by the element. Employment of an attribute on elements to the right in the table will supersede use on elements to the left. Generally, every cell has right and bottom borders, but only cells at the left side of the table have a left border and only cells across the top have a top border. Only one cell has four borders.

Table 4.18.3 Tabular Elements and Attributes (p = potential)

Attributes	Elements				Values
	`tabular`	`col`	`row`	`cell`	* = default
`top`	×	×			none*, minor, medium, major
`left`	×		×		none*, minor, medium, major
`bottom`	×		×	×	none*, minor, medium, major
`right`	×	×		×	none*, minor, medium, major
`halign`	×	×	×	×	left*, center, right, justify
`halign`		p			decimal, character
`header`			×		yes/vertical/no*
`row-headers`	×				yes/no*
`valign`	×		×		top, middle*, bottom
`colspan`				×	1*, positive integer
`width`		×			percentage
colors	p	p	p	p	

4.19 Program Listings

A `<listing>` is really a specialized type of `<figure>`, whose purpose is to hold computer code. Just like a figure, it has a `<caption>` and `<title>` which behave identically. However, the enclosed planar content is limited to a `<program>` or `<console>` (see Section 4.15).

4.20 Named Lists

As mentioned above, it is not possible to have a list be the target of a cross-reference. Should an entire list be *so important* that you need to point to it from elsewhere, then make it a **named list** by wrapping it in the `<list>` tag.

This element can begin with an optional `<introduction>`, then has a single, required list, which may be any of the three types. It concludes with an optional `<conclusion>`. It can have an `@xml:id` attribute, which in a way is the whole *raison d'être* for this construction. It will be numbered when rendered, and so also requires a `<title>`. You might think of this as similar to a `<table>`—bits of information organized spatially, via indentation and line breaks.

Since this element associates a number, title, to an entire list, we call it a "named list". What should we call a list that is authored within a paragraph and cannot be the target of a cross-reference? We call it an **anonymous list** when we want to make the distinction.

4.21 Sage

Until we can expand this section, get some ideas from Section 3.17. We will also collect a few items here, to be cleaned-up later.

For online output formats, sometimes the output of a Sage command can be overwhelming, and a bit complicated to parse. Many objects in Sage also have a LaTeX representation, which can be used to create a superior output format (for some purposes). Begin a cell with the "magic":

```
%display latex
```

Experiment with the following Sage code on the next line

```
integral(x^9*cos(x), x)
```

Boom! Very nice. Try replacing `latex` with `None`, `plain`, `ascii_art`, or `unicode_art`.

4.21.1 Sage Cell Server Design

The ability to execute, and edit, chunks of Sage code is provided by a distinct project, the Sage Cell Server[1]. Simplifying somewhat, the Sage code a reader sees (or has edited!) is shipped out to a running instance of Sage (on a server *somewhere*) and the code is executed there. The results of that computation are shipped back to the reader for display below the code.

Two implications of this design are

- It is not within your power to add additional packages for the supported languages.
- You cannot read a (data) file hosted on your project's site.

Fortunately, there are workarounds.

If your code needs a Python package, or an R package, or similar, and it is a standard open source package, then make a request on the Sage Cell[2] Google Group. Likely, it can be added/installed.

Unfortunately, the ability to read files *anywhere* on the internet was abused, so this capability had to be restricted to a finite list of servers. These include DropBox[3] and GitHub[4] where you might find it convenient to place files supporting your code. Note that for GitHub, you likely want to use a URL which is a "raw" file such as for the PreTeXt repository README[5] file, written with Markdown.

4.22 Interactives

TODO: until then examine copious examples in the sample article.

4.23 Side-by-Side Panels

Documents, pages, and screens tend to run vertically from top to bottom. But sometimes you want to control elements laid out horizontally. A `<sidebyside>` is designed to play this role. It is best thought of as a container, enclosing **panels**, and specifying their layout. Examples include three images, all the same size and equally spaced. Or a poem occupying two-thirds of the available width, with commentary adjacent in the remaining third. Or an image next to a table. But the most common use may be a single image (with no caption, and hence no number), whose width and horizontal placement are controlled by the layout.

See the schema for the exact items that are allowed in a `<sidebyside>`. To author, just place these items within `<sidebyside>` in the order they should appear, left to right. Then you add attributes to the `<sidebyside>` element to affect placement.

Instead of placing a `@width` attribute on each item, instead place this on the `<sidebyside>` element. A single `@width` will use the same value for each panel. For different widths, use the plural form `@widths` and provide a space-separated list of percentages. The default is to give each panel the same width, and as large as possible, which will result in no gap between panels.

The margins can be specified with the `@margins` attribute, which if given as a single percentage will be used for both the left and right sides. You may also specify asymmetric left and right margins with two percentages, separated by a space, in the same attribute. An additional option is to use the value `auto` which will set each margin to half of the (common) space between panels. This is also the default. In the case of a single panel, the left margin, right margin, and panel width should all add up to 100%.

Once the widths and margins are known, any additional available width is used to create a common distance separating panels. (Which is not possible when there is just a single panel.)

Independent of horizontal positioning, individual panels may be aligned vertically. The attribute is `@valigns` and its value is a space-separated list of `top`, `middle`, and `bottom`. The singular version, `@valign`, is used to give every panel the same alignment, using the same keywords. The default is to have every panel at the `top`.

[1] `sagecell.sagemath.org/`
[2] `groups.google.com/g/sage-cell`
[3] `www.dropbox.com`
[4] `github.com`
[5] `github.com/PreTeXtBook/pretext/raw/master/README.md`

We could give lots of examples, but instead it might be best to just experiment. Error-checking is very robust, so it is hard to get it too wrong. OK, we will do just one to help explain. Suppose a `<sidebyside>` contains three panels and has layout parameters given by

```
<sidebyside widths="20% 40% 25%" margins="auto" valign="middle">
```

Then there will be 15% of the width left to space out the panels. The two gaps are each 5% of the width, and the remaining 5% is split between the margins at 2.5% each. And the vertical midlines of each panel are all aligned.

For a single panel with no attributes, the panel will occupy 100% of the width. A single panel with a specified width will get equal (auto) margins, resulting in a centered panel.

Captioned items as panels deserve special mention. These will continue to be numbered consecutively, with one exception. If you place a `<sidebyside>` inside of a `<figure>`, then the `<figure>` will be numbered, and the captioned items inside the `<sidebyside>` will be **sub-captioned**. In other words, the second captioned panel of a `<sidebyside>` inside Figure 5.2 would be referenced as Figure 5.2.b.

An `<sbsgroup>` ("side-by-side group") contains only `<sidebyside>`, which are displayed in order. However, all of the layout parameters allowed on a `<sidebyside>` may be used on an `<sbsgroup>`. This might allow a collection of fifteen images to be laid out in three rows of five images each, with widths and spacing identical for each row because the parameters are specified on the `<sbsgroup>` element. In this way, simple grids can be constructed. Note that any layout parameters given on an enclosed `<sidebyside>` will take priority over those given on the `<sbsgroup>`. Captioning behavior extends to an entire `<sbsgroup>`.

Since `<sidebyside>` and `<sbsgroup>` are containers they cannot be referenced and so do not have an `@xml:id`. However, you can reference their individual contents if they are captioned, and you can reference an enclosing `<figure>`.

Generally, a `<sidebyside>` or `<sbsgroup>` can be placed as a child of a division, or within various blocks, such as `<proof>` for example. See the schema for (evolving) specifics.

It should be clear now that a `<sidebyside>` is more about presentation than most PreTeXt elements, though there is some semantic information being conveyed by grouping the panels with one another.

4.24 Front Matter

A single `<frontmatter>` element can be placed early in your `<book>` or `<article>`, after some metadata, such as the overall `<title>`. It is optional, but likely highly desirable. The following subsections describe the items that may be employed within the `<frontmatter>`. Most are optional, and some may be repeated. An `<article>` differs in that it must contain a `<titlepage>` and then may *only* contain an `<abstract>`. Generally, these will get default titles, localized in the language of your document, but these defaults may also be replaced by giving a `<title>` element. None of these divisions themselves is numbered, precluding any content within that is numbered. So, for example, no `<figure>` may be included. But you could choose to include an `<image>`, perhaps within a `<biography>`.

If a component of the front matter cannot be numbered, how best to subdivide something like a `<preface>`? This is a good use of the `<paragraphs>` element. It allows for a (minimal) title, but cannot be subdivided further. See the later part of Section 4.6 for more about this exceptional element.

These elements must appear in your source in the order given below, and will appear in your ouput in the same order, which is a generally accepted order used in the production of books. So, for example, even if you author an `<acknowledgement>` between two `<preface>`, your output may (will?) place the Acknowledgement before the first Preface.

(We have not yet described the contents of these various elements in full detail.)

4.24.1 (*) Title Page

Required. Since the entire `<frontmatter>` is optional, we assume that the front matter at least includes the appearance of the document's overall `<title>`.

4.24.2 (*) Abstract

Optional, and only available for an `<article>`.

4.24.3 (*) Colophon

The *front* colophon. (There is also a *back* colophon, see Subsection 4.25.6). Sometimes this is also called the **copyright page**.

4.24.4 (*) Biographies

Multiple `<biography>` elements, one per author.

4.24.5 (*) Dedication

A single `<dedication>` element, that might include multiple dedications (perhaps by different authors).

4.24.6 (*) Acknowledgements

A single `<acknowledgement>` element (note spelling), that becomes a division, and so can contain paragraphs, lists, etc. The Chicago Manual of Style [1, 1.52] suggests that if these are short, they may be contained in a preface.

4.24.7 (*) Forewords

As of 2021-07-16 the `<foreword>` element is not fully implemented. Please make a feature request if you need it.

A `<foreword>` is written by somebody other than the author. The name of the writer of the foreword need to be included—at the end is a good location.

4.24.8 (*) Prefaces

Multiple prefaces are a distinct possibility, and in this case providing a different `<title>` for each would be essential. Examples might include: "Preface to the Third Edition", "How to Use this Book", or "To the Student". More ad-hoc material, such as a translator's note, can be handled as a preface.

Best Practice 4.24.1 Understand the Role of a Preface. *Chicago Manual of Style* [1, 1.49] begins with "Material normally contained in an author's preface includes reasons for undertaking the work, method of research, ..." Note that a preface is not introductory content and is not an introduction. It is written from the author's point-of-view, and may include information about why they are qualified to write on the topic of the book. If there are several editions, the prefaces to the newer editions are placed first. See the related Best Practice **??**.

4.25 (*) Back Matter

4.25.1 (*) Appendices

Automatic lists (Section 4.28) can appear anywhere, but an appendix is a very natural place to place one.

4.25.2 (*) Glossary

4.25.3 (*) References

4.25.4 (*) Solutions

4.25.5 (*) Index

4.25.6 (*) Colophon

The *back* colophon, what most authors think of as *the* colophon. (There is also a *front* colophon, see Subsection 4.24.3).

4.26 Index

Continuing our basic discussion from Section 3.23, we discuss some details of making and using index entries. We will begin with how you *procedurally* author an index entry with PreTeXt syntax, and then move to general principles about how to use these constructions to create an effective index. So these two subsections are intimately linked.

4.26.1 Syntax and Placement of Index Entries

Best Practice 4.26.1 Capitalization of Index Entries. The headings (entries) of an index are authored entirely in lower-case, unless it is a proper noun (name, place, etc.) which would normally be capitalized in the middle of a sentence. We are not able to provide any enforcement of this advice, nor any assistance. It is the author's responsibility to provide quality source material in this regard. We do sort entries so that an entry with an initial capital letter arrives at the right location in the index.

Where you place an `<idx>` entry is critical. With LaTeX output, you will get the traditional page number as a locator in your index. With HTML output we can be more careful. We will look to see which sort of structure contains the `<idx>`. Maybe it is an `<example>` or a `<subsection>`. If so, the index will contain a locator that is a knowl of the example, or a link to the subsection. The distinction is the size of the object, we do not knowl divisions. The exception is a paragraph (`<p>`) that is a child of a division, and then the locator is a knowl of the entire paragraph. Remember that a knowl contains an "in-context" link which can take the reader to the original location of the content in the knowl.

A lot happens in a PreTeXt paragraph, especially when producing HTML. Sometimes an `<idx>` can get in the way. Our recommendation is to put `<idx>` entries *between* sentences, and not at the start or end of the paragraph. They can be authored with each on their own line. If you do not need the specificity of a paragraph, then locate the appropriate structure and author the `<idx>` right after the `<title>` (or where one would be).

A **cross-reference** in an index is a pointer to another index entry. These are rendered as "See" and "See also." You can add `<see>` and `<seealso>` elements within an `<idx>`, so long as it is structured with `<h>`. Then it is placed after the last `<h>`. A "see" cross-reference is a direct pointer to another entry in the index. It cannot have a locator as well. When you build the HTML output, we will recognize this situation and produce a warning. A "see also" cross-reference is an additional pointer, and so it must have a locator to go with it (you will author two `<idx>` with identical headings, the first without a `<seealso>` to create the locator, the second with the `<seealso>` to create the cross-reference. Again, when you build the HTML output, we will recognize a `<seealso>` without a locator and produce a warning.

Follow these directions and PreTeXt will format cross-references for you, in the style suggested by the *Chicago Manual of Style* [1] for HTML output, and according to LaTeX's style for print and PDF.

(2019-03-04) We have consciously not said anything specific about what to place inside a `<see>` or `<seealso>` element. At this writing, you need to supply the text. Of course, this is error-prone and you will need to consult CMOS for formatting guidance. But we have plans to do this the PreTeXt way. First, the `ref/xml:id` mechanism will be used to automatically create the correct text for the cross-reference, both content and format. Second, these will become live links in electronic formats.

Certain index entries do not sort very well, especially entries that begin with mathematical notation. Our first advice is to avoid this situation, but sometimes it is necessary. The `@sortby` attribute on an `<h>` element can contain simple text that will be used to override the content shown to the reader during the sorting of the index.

4.26.2 Advice on Indexing

An index is a navigational aid for your readers (and you). We do not assume that a reader remembers where anything is, nor that the Table of Contents is a replacement for part of the index. Some readers of the index may not have even read your book yet, and are looking to get a feel for the range of topics as part of the decision of whether or not to read your book at all, or if it will be useful to have. It should be comprehensive, including everything substantive.

Indexing is a job for a skilled professional, and most authors produce poor indexes. The tips in this section will help you avoid the most common pitfalls. We follow recommendations from the *Chicago Manual of Style* [1, Chapter 18], *Indexing for Editors and Authors: A Practical Guide to Understanding Indexes* [3], and Pilar Wyman of Wyman

Indexing[1].

Terminology. The basic element of an index is an **entry**, which consists of a piece of information and its **locator**. For example:

```
normal subgroup,  37
```

is an entry indicating that information about "normal subgroup" can be found on page 37. Indexes are (usually) organized alphabetically, with a **main heading** aligned with the left margin, and progressively indented **subheadings** below the main heading.

Often it is desirable to place the same locator under more than one heading, known as **double posting**. For example, a desirable addition to the sample entry above is

```
subgroup,  28
   normal,  37 .
```

An alternative to double posting is **cross referencing**, using *see* and *see also*. Typically cross references are used to avoid repeating a large number of entries, or to direct the reader to related topics.

An index may start with a **headnote** giving advice about using the index. Typically a headnote is not necessary unless the index has some unusual features.

Basic principles. The purpose of an index is to point the reader to information. Point to, not repeat. For example, acronyms should be indexed at the location where they are defined, not at every place they appear, and it is not necessary to define the acronym within the index. People and places should be indexed when information is given about them, not every time they are mentioned.

A good index has multiple ways to find the same information. Being redundant is desirable, because it increases the chance the reader finds what they seek in the first place they look.

Indexing is best done after the text has been written. Adding index entries while writing the text may seem to be a labor-saving device, but if you are not an experienced indexer, those entries will only be a small fraction of the final index.

Topics should be indexed in multiple ways. If a term is defined, you should also think of other words the reader might search for. For example, you may define "limit point" and consistently use only that term, but an index entry for "accumulation point" with a "*see* limit point" locator would be appropriate.

Use **disambiguation** to distinguish identical terms with different meanings. For example

```
isomorphism (of groups),  55
isomorphism (of rings),  123
```

Both of those entries should also be double posted under the main headings of "group" and "ring", respectively. No disambiguation is needed for those entries.

Singular or plural forms of nouns should reflect the language in the text. So if a chapter is titled "Mammals", then use a heading `mammals`. And if the chapter is titled "The Mammal Class", then use a heading `mammal`.

An index is typically as long as 5% of the main text. With many figures, or other structures creating additional whitespace, the percentage may be lower. If your primary output is online, length may not be an issue. For print, there are strategies for pruning an index.

Once you have finished the text, and then finished the index, it is time for a thorough review of the index. There will be places for consolidation, often due to using variants of particular words. You may wish to remove subheadings which all appear within the range given in the heading. For example,

```
fish, 204-212
   bass, 208-209
   salmon, 210
   trout, 207
```

could have all of its subheadings removed, especially if space is an issue.

[1] `www.wymanindexing.com/`

Common pitfalls. Sometimes it takes less than one second to determine that an index is poor. If a quick glance reveals that the index consists mostly of main headings with very few subheadings, then few readers will find it to be useful. Double posting, which may mean more than literally two entries with the same locator, will help readers find what they are looking for. Most of those entries will be in subheadings.

Another instantly recognizable problem is too many locators in one entry. This entry

```
asymptote,  37, 48
```

is probably fine. But once you have three or more locators in an entry, then your index may be improved by adding some subheadings. If the locators in the above example refer separately to "horizontal" and "vertical", then probably two subheadings would be more useful than two undifferentiated locators in one entry.

An additional problem which can be seen at a glance if you know what to look for, is the absence of any main headings with a large number of subheadings. On almost any subject there are topics which are addressed repeatedly. This should be reflected in the structure of the index. For example, in a group theory textbook there should be several entries under "group, examples". In an introductory calculus book the index should help the reader locate the derivative of many different elementary functions.

Index headings should be nouns, not adjectives. An adjective may be important, and you should use it, but it should not be the entire content of a heading since it is not an idea by itself. But it may be a subheading. For example, suppose you have a paragraph on "highland sheep." Then *both* of the following should appear in your index, since a reader might consult both locations.

```
highland sheep, 45
```

```
sheep
   highland, 45
```

4.27 Notation

We continue the introduction at Section 3.23. A notation list, like an index, is a specialized collection of cross-references. So some of the philosophy here applies equally well to the `<idx>` and `<index-list>` elements, and vice-versa. (See Section 4.26.)

To generate a list of notation employed in a book or article, use the `<notation-list/>` element. This empty element belongs in an `<appendix>`. Likely it is the only content, or you might include some preliminary material. The title of the `<appendix>` is up to you and is not automatic.

Some authors like to make definitions inside of paragraphs, ideally using a `<term>` element. This is a natural place for a `<notation>` element. So this approach gives an author a lot of flexibility in location.

Other authors like to make definitions using the `<definition>` element, since it creates a heading and number, allows a `<title>`, and can easily serve as the target of a cross-reference. So this is another good place for a `<notation>` element. But now, associate it clearly with the `<defintion>` by placing it in the metadata, early on, after the `<title>`. And not in some subsequent paragraph. The reason will be clear in just a bit.

How is a `<notation>` element constructed? It has two elements. The `<usage>` should be a sample piece of mathematics using the necessary symbols, and wrapped in a *single* `<m>` element. The second element is `<description>` and should be a short phrase, or sentence-like material, decoding the sample usage, and may include `<m>` elements. The reader sees nothing in the output at the location of the `<notation>` element.

The automatically-generated notation list is then a three-column table, in the order of appearance, with the sample usage, the description, and a **locator**. For output derived from LaTeX, such as print or PDF, the locator will be the page number of wherever you placed the `<notation>` element. For HTML the locator is much better—it is a knowl, for either a paragraph or for an entire definition. The latter possibility explains why it is better to place the `<notation>` element inside a `<definition>`, if possible, rather than in a paragraph that is a constituent of a `<definition>`.

4.28 Automatic Lists

Sometimes it is useful to have an automatic list of various elements of one kind in a book, other than the ones already available in a PreTeXt document. The predefined ones include an index (see Section 4.26) and a list of notation (see Section 4.27). Examples of lists one might wish to create could include lists figures, computational listings, or theorems.

There is a very flexible way to make a list of various blocks (or perhaps other items) in your text. Use an empty `<list-of/>` element as a child of a division. A very natural use would be to create an `<appendix>` for the sole purpose of holding one such list. This is why this feature is frequently used in the back matter. But you could place an **automatic list** many other places.

We will illustrate with an example. Suppose you know your book has theoretical results only in `<theorem>` and `<lemma>` elements. So, for example, you never use `<corollary>` elements. Then you could author

```
<appendix xml:id="appendix-list-results">
  <title>List of Results</title>

  <list-of elements="theorem lemma" divisions="chapter" empty="yes"/>
</appendix>
```

The result will be a link to every `<theorem>` and `<lemma>` in the entire book, using a clickable with its type, number and title. See Appendix **??** for an example. In HTML output the clickables will usually be knowls, which is especially handy. The list will be organized with the titles of the chapters as headings. The `@divisions` attribute can have several types of divisions listed, such as both `<chapter>` *and* `<section>`. The `@empty` attribute set to `yes` indicates that a division heading should be used even if there is nothing on the list contained within. The default for `@empty` is `no`.

This feature is best used when the items in the list have been authored with titles, which greatly increases the utility of the list for your reader. Review Best Practice 4.8.1 if this advice is new to you.

There is a `@scope` attribute, which should be the name of an element which is a division containing the location of the `<list-of>` element. Then the list is restricted to items within the specified division. For example, if you have the `<list-of>` inside a `<subsection>` built for this purpose, and you use `@scope="section"` then the list will have all the items from throughout the section containing the list.

There are four types of exercises, based on their location: inline, divisional, worksheet, and reading questions. These may be specified inside `@elements` by the **pseudo-elements**: '`inlineexercise`', '`divisionexercise`', '`worksheetexercise`', '`readingquestion`'. (These are just strings meant for this purpose, and are not *real* PreTeXt elements.)

There may be an argument for a `@ref` attribute that would behave similar to `@scope`. Make a feature request if you need it?

4.29 URLs and External References

4.29.1 URLs to External Web Pages

The `<url>` element is used to point to *external* web pages, or other online resources (as distinct from other internal portions of your current document, which is accomplished with the `<xref/>` element, Section 3.4). The `@href` attribute is always necessary, as it contains the full and complete address of the external page or resource. Include everything the URL needs, such as the protocol, since this will be most reliable, and as you will see it never needs to be visible. The element always allows, and then employs, a `@visual` attribute for a provided more-friendly version of the address. Finally, the content of the element, which becomes the clickable text in electronic formats, can be authored with the full range of PreTeXt markup generally available in a title or sentence. A typical use might look like

```
    <url href="https://example.com/" visual="example.com">Demo Site</url>
```

This will render as Demo Site[1]. Note the automatic footnote providing the visual version in a monospace font. If a `<url>` has content, and no `@visual` attribute is given, then the `@href` will be placed in a footnote, though there will be an attempt to remove standard protocols. Compare

```
    <url href="https://example.com/">Demo Site</url>
```

[1]`example.com`

which will render as Demo Site[2] versus

```
<url href="mailto:nobody@example.com">Bouncing Email</url>
```

which will render as Bouncing Email[3].

If you do not provide any content for a `<url/>` element, then the clickable text will be the actual URL with a preference for the (optional) `@visual` attribute, rather than the mandatory `@href` attribute. This should be considered as disruptive to the flow of your text, and so a poor alternative to the content version just discussed (see Best Practice 4.29.1). But it might be a good choice in something like a list of interesting web sites. Whether or not a simplified version of the address, via the `@visual` attribute, is desirable will depend on the application. As an example, using the optional `@visual` attribute we have

```
<url href="https://example.com/" visual="example.com"/>
```

This will render as `example.com`. Note that there is no footnote since the visual version is already apparent.

If you want to squelch the automatic footnote on a `<url>` element with content, you can explicitly set the `@visual` attribute to an empty string as `visual=""`. This signal will inhibit the automatic footnote. This should be a *very rare* occurence, since you are denying readers of some formats from seeing even a hint of the actual URL.

An extreme example of this behavior is a regular footnote which contains a URL. Because an automatic footnote, inside another footnote, becomes problematic in some conversions, we squelch the footnote-within-a-footnote. A best practice here is to just list nearby a URL, likely using the `<c>` element to get a monospace font.

A `<url>` inside a `<title>` has been accounted for, but should be used with caution.

As with the rest of PreTeXt we have taken care to handle all of the exceptional characters that might arise in a `<url>`. So author normally, using the necessary keyboard characters, only taking care with the two XML characters, `<` and `&`, which need escaping (see Section 3.14). Use **percent-encoding** (aka **URL encoding**) for the `@href` attribute, if necesary, to include special characters, such as spaces. See Subsection 4.29.4 below for a common need for the ampersand character, and a further caution about percent-encoding of URLs.

Finally, for conversion to LaTeX/PDF output it gets extremely tricky to handle all the various meanings of certain escape characters in URLs in more complicated contexts (such as tables, footnotes, and titles), so there may be some special cases where the formatting is off or you get an error when compiling your LaTeX. We have anticipated most of these situations, but we always appreciate reports of missed cases.

4.29.2 Data URLs

A `<dataurl>` element is very similar to the `<url>` element just described. The purpose is to point to an actual file that will be of use to your readers. What actual happens when a reader clicks on it is dependent on the format of the PreTeXt output and that reader's environment. Maybe the file will be downloaded, or maybe a particular application will open the file. That part is out of our hands. Use an `@href` attribute in the same way as for `<url>`, and the content and the `@visual` attribute also behave similarly.

The one key differerence is that you can also use a `@source` attribute in place of `@href` and point to a file that you provide as part of your project (not unlike providing a photographic image via the `<image>` element). Place the file in your collection of external files (see Section **??**) and provide the path to your file from below the directory of external files in the `@source` attribute. For HTML output, PreTeXt will do the rest. For more static formats, you can set a base URL (see Subsection **??**) and you will get a complete URL that points to the instance of your file hosted with the rest of your HTML output.

Notice that this element provides limited functionality, at best just a hyperlink to a file. For data files that you want a reader's in-browser computer program to process, read about the `<datafile>` element at Section 4.16.

4.29.3 Visual URLs

By a **visual URL** we mean a version of a URL that is simpler than the "real" URL, but that provides enough information that a reader can type the URL into some other device with a minimum of effort, and with success. Consider that your project may someday be a print (hardcopy) book, or that your project will be converted to braille for a blind reader. These are some ideas about making a URL simpler. We welcome more ideas.

[2] `example.com/`
[3] `mailto:nobody@example.com`

- Remove standard/default protocols like `http://` and `https://` which most browsers will furnish in their absence.
- Sites like StackExchange[4] list posts with a long identifying number, followed by something that looks like the title. In practice, the number is enough.
- Experiment with dropping a trailing slash—they are frequently unnecessary.
- Often a leading `www.` in a domain name is not necessary.
- Try providing just a domain name in place of a top-level landing page, it will often redirect to a longer URL.
- You could use a URL shortener[5], though some thought should be given to its longevity[6]. Will you remember where your short URLs point once they are no longer functional. Safer to have your long URLs in an `@href` in your source, and use PreTeXt to make them friendlier.

Best Practice 4.29.1 Craft URLs Carefully. Your writing will be smoother, and easier on your readers, if you do not interrupt a sentence with a long URL, unless somehow it is really of interest and relevant right there. So provide content (the "clickable" text) when you use the `<url>` element (rather than an empty `<url/>`). This obligates you to provide a `@visual` attribute, which feels a little like a tedious exercise. But this will be very welcome to some of your readers, those who are unable or prefer not to use electronic formats. Just above (Subsection 4.29.3), we provide suggestions for crafting these to be more pleasing, but still useful, versions of URLs.

4.29.4 Characters in URLs

A URL can have a **query string**, which has a list of parameters following a question-mark. The parameters are separated by ampersands (&), which will need to be escaped, so as to not confuse the XML processor. So use `&amp;` anywhere the ampersand *character* is necessary, such as a `@source` attribute, or a monospace version of a URL achieved with a `<c>` element. Also, the question-mark character should *not* be URL-encoded (`%3F`) (despite advice just given above), so if necessary edit it to be the actual character. General advice about exceptional characters in XML source can be found in Section 3.14.

4.30 Video

A video is a natural way to enhance a document when rendered in an electronic format, such as HTML web pages. It might be additional information that is hard to communicate with text (marine invertebrates swimming), a lecture or presentation that augments your text, or even some artistic work, such as a symphony legally hosted on YouTube, when you could never hope to get copyright clearance yourself.

PreTeXt supports videos you own and distribute with your source, videos shared openly on the Internet via stable URLs, and videos available on YouTube. Go straight to to the end of this section to see how easy it is to incorporate a YouTube video.

HTML5 web browsers are able to play video files in three formats, summarized in the following table.

Table 4.30.1 HTML5 video formats

Format	Extension	Reference
Ogg, Theora	`.ogg`	Free and open, Wikipedia[1]
WebM	`.webm`	Royalty-free, Wikipedia[2]
MPEG-4	`.mp4`	Patent encumbered, Wikipedia[3]

[4] `stackexchange.com`
[5] `https://w.wiki/4QA`
[6] `https://w.wiki/4eEM`
[1] `en.wikipedia.org/wiki/Ogg`
[2] `en.wikipedia.org/wiki/WebM`
[3] `en.wikipedia.org/wiki/MPEG-4_Part_14`

4.30.1 Video Element

The `<video>` element is used to embed a video in output formed from HTML. Subsections below describe the different ways to indicate the source of the video. The video may be placed inside a `<figure>` or can be a panel of a `<sidebyside>`. The former will have a caption, be numbered, and hence can be the target of a cross-reference (`<xref>`). The latter is anonymous, but allows for horizontal layout, and combinations with other panels.

Size is controlled by a `@width` attribute expressed as a percentage (on the `<video>` element when used in a figure, or as part of the `<sidebyside>` layout parameters). Height is controlled by giving the **aspect ratio** with the `@aspect` attribute on the `<video>` element. The value can be a ratio expressed like `4:3` or a decimal number computed from the width divided by the height, such as `1.333`. The default for videos is a 16:9 aspect ratio, which is very common, so you may not need to specify this attribute.

Options include specifying a `@start` and an `@end` in seconds as integers (no units) if you only want to highlight a key portion of a video. The `@play-at` attribute can take the following values

`embed` Play in place (the default action).

`popout` Play in new window or tab, at 150% width.

`select` Provide the reader the choice of the other two options.

In an educational setting, sometimes the preview images provided by YouTube can be distracting, or for an author-provided video you may wish to provide your own preview image. The `@preview` attribute can take on the following values

`generic` PreTeXt supplies a Play-button image.

`default` Whatever the video playback provides. This is identical to simply not including `@attribute` at all

Path to an image file
Typically, this will be a relative path, starting with `images/`. This image will be used as preview for the online version and the print version.

4.30.2 Author-Provided Videos

If you own and possess your video content, then you can distribute it with your PreTeXt source, and it can be hosted as part of your HTML output. Then the `@source` should be a relative file name that points to the file containing the video. If you are able to provide more than one of the three formats in Table 4.30.1, then you can provide the filename *without an extension*. If a browser cannot play one format, it may be able to play another. PreTeXt will write the code to make that happen, preferentially in the order of the table (more open formats first!). In other words, you can provide files in more than one format and increase the likelihood that a reader's browser will find a format it can playback.

4.30.3 Network-Hosted Videos

If a video is shared openly on the Internet, you can simply provide the full URL for the `@source` attribute. All the other attributes are the same as for the author-provided case, above. Read Subsection 4.29.4 for some considerations when authoring a URL, since there are a few gotchas.

You can frequently discover the URL of a video by first playing it, and then using a context menu (e.g. via a right-click) to reveal an option to copy the video's location. However, note that there are various techniques sites use to make such a URL temporary, or otherwise unusable. So do some research about potential uses and test carefully. Our example below is provided from a United States government site.