Interactive content using TEX4ht

Richard Koch

Abstract

TEX4ht converts LATEX source into web pages. This article explains how to add interactive content to these pages, using TEX4ht and straightforward copying from web sources. The techniques should work on all computer platforms. Some refinements to the TEX4ht methods are also discussed.

1 Introduction

Let me begin with three vignettes.

I started attending TUG conferences in 2001, and along with expected talks there were a few surprises. In 2005, an expert from England predicted that TEX would survive for four more years and then be replaced. He was teaching in the Open University system where students work remotely, and he wanted to include interactive content in his lectures. I thought the talk was nonsense. Then COVID hit.

The 2004 Practical TEX conference was held at Fisherman's Wharf in San Francisco, and included a talk by Ernest Prabhakar, an Apple engineer. After that talk, Prabhakar met with Mac users and others including Hans Hagen, all sitting around a large conference table. Hans was trying to convince Apple to allow Java programs to run in their pdf viewer so interactive elements could be added. I sat next to Prabhakar and got to see how he operates. He was fully engaged in the conversation, but simultaneously he was surfing the web — the fastest surfer I have ever seen. Eventually he said to Hans, "It appears to me that you are the only one in the world writing Java in pdf files."

I'm one of those users who updates TEX Live daily while drinking my morning coffee. Sometime in 2022 I noticed that `tex4ht` was on every day's update list. So I wrote the TEX Live mailing list asking that this bug be fixed. To my surprise, I was told that the updates were genuine; Michal Hoftich, who maintains TEX4ht, makes updates almost daily.

2 PDF and HTML fifteen years later

The Fisherman's Wharf conference was 18 years ago, and some issues are clearer with the passage of time. Today every computer platform has excellent software to display pdf files, and every computer platform has an up-to-date web browser. It seems clear that pdf is the right format for static documents, and that html is the right format for documents with interactive content. Other formats may emerge, but that will only happen if an activity cannot be supported by pdf or html. (Although pdf has facilities for interactivity, they are rather infrequently used compared to interactive html.)

3 A TeXShop detour

I wrote TeXShop, a front end for TEX on the Macintosh. TeXShop is relevant here only because it explains how I was led to reexamine TEX4ht.

Typesetting in TeXShop is controlled by "engine files", small shell scripts that users can edit which call TEX binaries. After typesetting, an engine searches the source directory for a pdf file with the same name as the source and opens it in a pdf preview window if found. This August I added code which searches for an html file with the same name as the source, and opens it in an active web window if found. These windows are created using the Cocoa programming APIs, so they are part of TeXShop rather than external Mac applications like Preview and Safari.

With this change, it is easy to "typeset" html files. Similarly, it is easy to support PreTEXt, a project where authors write xml source and then convert the source to pdf, html, and other formats.

So it was natural to try TEX4ht, which accepts a LATEX source file and outputs html (among other things). TeXShop now has a typesetting engine which typesets the source twice, once with TEX4ht and once with pdflatex. The TEX4ht output is opened in a web viewer and the pdf output is opened in a pdf preview.

I had seen demonstrations of TEX4ht given by Eitan Gurari, the original author of TEX4ht. Indeed at that 2004 conference at Fisherman's Wharf, Prabhakar's talk was immediately followed by a talk by Gurari on TEX4ht. At the time, TEX4ht was outputting mathematics using pictures, and the results were a little crude.

In the years since then, MathML was invented, and then MathJax was created and provided beautiful rendering of MathML code. TEX4ht adopted these technologies.

I selected a 20-page set of lecture notes, with extensive mathematical equations and many illustrations. The document used `hyperref`, `amsmath`, and other packages. I typeset it with TEX4ht, producing html. Typesetting was fast — and the html output was amazing! The mathematical equations were crisp and clear, the illustrations were fine; to tell the truth, I doubted that I was seeing html. As a test, I resized both windows. The text in the pdf window shrank since the pdf had been configured to "fit in window". The text in the html window reflowed.

doi.org/10.47397/tb/43-3/tb135koch-tex4ht

4 Interactive content

TEX4ht therefore allows you to convert old and new LATEX static documents into web documents. But can you add interactive content to these documents? Yes, as this article will demonstrate.

Select an old document you have lying around the house. You'll be able to add interaction to it by the end of the next two sections. Don't typeset immediately because a couple of steps are needed. Both are given in these two sections.

First we need a method to write source code which will only appear in the html version of the document. The following code does the trick:

```
\ifx\HCode\undefined
  % source for pdf document
\else
  <!-- source for html document -->
\fi
```

The `\HCode` tested here is a command that appears only in TEX4ht. Some web documents recommend the `ifpdf` package, but that fails when typesetting with XETEX.

Next, we need to switch from writing LATEX code to writing html code which TEX4ht will insert verbatim into the final document without processing. The following code suffices:

```
\ifx\HCode\undefined
  % source for pdf document
\else
  Initial words for html document.
  \begin{html}
     <!-- direct html input -->
  \end{html}
\fi
```

Finally we need something interactive. We'll use a piece of SageMath code, which is explained in a later section. Putting all this together, add the following lines to your document, creating a new section in the web version.

```
\ifx\HCode\undefined
\else
 \section{An Experiment}
 \begin{html}
  <div class="compute">
  <script type="text/x-sage">
    plot(sin(x), (x, 0, 2*pi))
  </script></div>
 \end{html}
\fi
```

Running this, we discover a minor problem. TEX4ht does not understand the command `\begin{html}`, and your web browser does not understand the lines calling Sage.

5 The header

To solve the problem, we must add the following header immediately after `\begin{document}`, before any other code is inserted. (The columns in *TUGboat* are narrow; the long `sagemath.org` url below needs to be on one line, and other source lines would usually be combined. Also, the source for this article is available from the article's web page.)

```
\ifx\HCode\undefined
\else
 % declare environment html:
 \ScriptEnv{html}
 {\ifvmode\IgnorePar\fi\EndP
   \NoFonts\hfill\break}
 {\EndNoFonts}
\fi

\ifx\HCode\undefined
\else
 % following url needs to be on one line, sorry:
 \begin{html}
  <script src="https://sagecell.sagemath.org/
static/embedded_sagecell.js">
  </script>
  <script>
  // Make div with id `mycell' being a Sage cell
  sagecell.makeSagecell({
    inputLocation:  '#mycell',
    template:  sagecell.templates.minimal,
    evalButtonText: 'Activate'});
  // Make div with class `compute' a Sage cell
  sagecell.makeSagecell({
    inputLocation: 'div.compute',
    evalButtonText: 'Evaluate'});
  </script>
 \end{html}
\fi
```

This header is divided into two parts, each preceded by an `\HCode` test so it is only active when typeset by TEX4ht. The first defines `\begin{html}` for TEX4ht. The second defines the Sage commands for the browser. Notice that the second command is also preceded by `\begin{html}` and thus is inserted directly into the final html document.

Now your source document has everything required for interaction, so go ahead and typeset it with TEX4ht. The recommended way to typeset is

```
make4ht sourcefile.tex "mathjax"
```

In the pdf version of the document, nothing changed. The html version will contain an additional section pictured below. Notice the button labeled "Evaluate" (fig. 1). When this button is pushed, the display changes to the form shown on the second image (fig. 2).

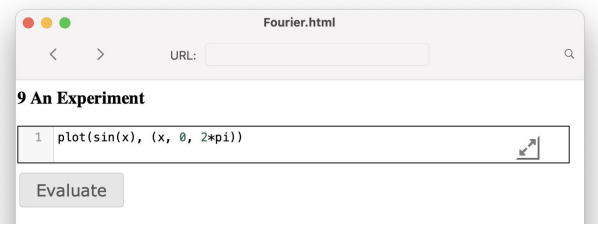


Figure 1: Sage Evaluate button and editable function.

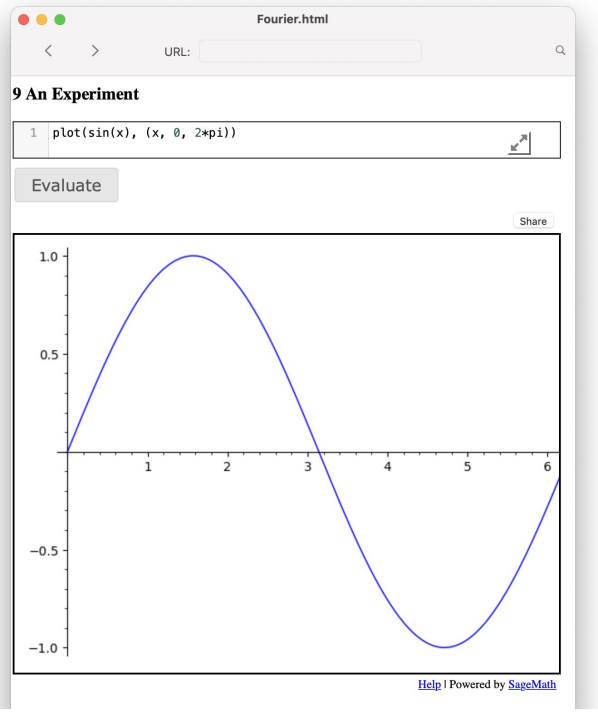


Figure 2: Original plot results.

But there's more. The SageMath code shown in both images is editable. If this entry is changed to

```
plot(log(x), (x, .1, 10))
```

a graph of the logarithm is plotted, and if it is changed to

```
plot(sin(x) + cos(3*x)/2, (x, 0, 2*pi))
```

an alternate periodic function is plotted. Therefore the four lines of code we added for Sage did not just provide a plot of the sine function. It provided a plotting machine which readers can use to plot any function!

6 SageMath

SageMath is an open source alternative to the computer algebra systems Magma, Maple, Mathematica, and MATLAB. The project was created by William Stein, a mathematician at the University of Washington, and first released on February 24, 2005. See `sagemath.org` and `wiki.sagemath.org`. Sage is mostly written in Python, but it integrates many previous open source projects written in C, Lisp, and Fortran. Among these are Gap, Macaulay, Maxima, Octave, and R. The program has been used for serious research on elliptic curves, finite groups, and many other areas, and has an active support group.

Although the Sage web site has install packages for major computer platforms, our use of Sage does not depend on installing SageMath, either for the author or for the reader on the web. Instead, Sage maintains a server which can run Sage over the web. See `https://sagecell.sagemath.org` and other links from that page for details.

Our previous Sage example contains a single line to plot a function. However, that line can be replaced by an arbitrary Sage program, which can be several pages long. We list several examples. All of these examples come from web pages at the Sage site; I have just copied and pasted code by others.

Here are two examples of calls to Sage:

```
\begin{html}
<div class="compute">
<script type="text/x-sage">
x, y = var('x,y')
plot3d(sin(x^2 - y^2), (x,-2, 2), (y,-2,2))
</script></div>
\end{html}
```

and

```
\begin{html}
<div class="compute">
<script type="text/x-sage">
u,v = var('u,v')
fx = (3+sin(v)+cos(u))*cos(2*v)
fy = (3+sin(v)+cos(u))*sin(2*v)
fz = sin(u)+2*cos(v)
parametric_plot3d([fx, fy, fz], (u, 0, 2*pi),
       (v, 0, 2*pi), frame=False, color="red")
</script></div>
\end{html}
```

The output from executing these commands is shown on the next page (fig. 3). However, this (pdf) article doesn't show the most amazing thing. If these objects are grabbed with the mouse, they rotate and magnify instantly in real time.

For the mathematically inclined, I'll show two more examples on the next page, without giving the Sage code, which can be found on various Sage web sites. Figure 4 illustrates numerical integration. The function can be set by the reader, the number of division points can be set, and the algorithm determining the top of each rectangle can use the value of the function at the left, right, or middle, or the maximum or minimum value. Since Sage can integrate symbolically, the exact value of the integral

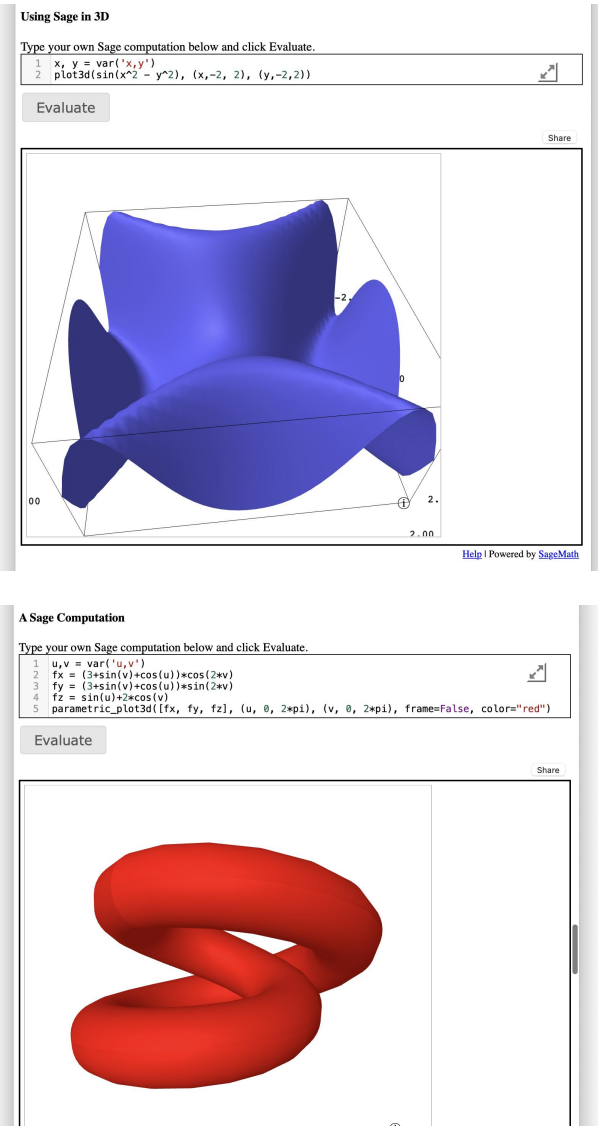


Figure 3: Two fancy plots, which in html output can be transformed in real time.

is shown at the bottom, and finally the numerical approximation is computed and shown.

In figure 5 the Taylor series of a function selected by the user is computed, and both the function and its approximation are plotted.

7 YouTube videos

Did you know that if you right-click while playing a YouTube video, a contextual menu appears allowing you to "copy embed code", which can be pasted into a web page?

I found a lecture by John Maynard, one of the four Field's Prize winners at the International Congress of Mathematicians for 2022. It is fun to watch this video for the depth and clarity of his

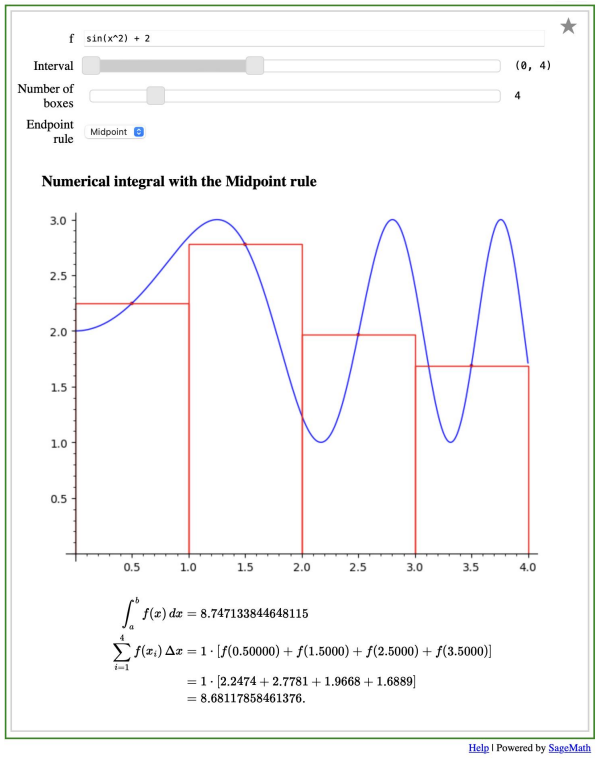


Figure 4: Numerical integration example with Sage.

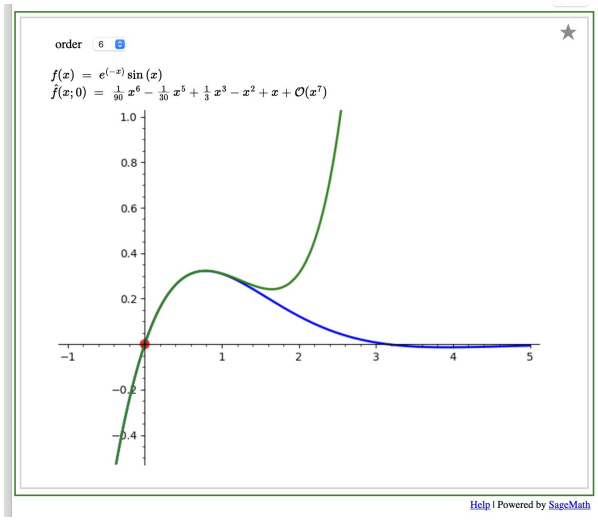


Figure 5: Taylor series of a user-selected function.

mathematics. I confess that I also watched because Maynard is left-handed and we lefties need to stick together. I don't understand a word of the code which YouTube provided when I clicked, but I added it to a TEX4ht source page and it worked. Figure 7 shows a frame of the video, and here is some of the source code, as copied from YouTube:

```
\begin{html}
```

Figure 6: Frame from YouTube video, playable live in html output.

```
<iframe width="928" height="522"
 src="https://www.youtube.com/embed/kQqBeuk_xQw"
 title="13. Large gaps between primes ..."
 frameborder="0"
 allow="accelerometer; autoplay; ..."
 allowfullscreen></iframe>
\end{html}
```

8 Mathematics

Often authors ask a question of readers and provide a multiple choice answer. If the reader answers correctly, they are told to go to the next section; otherwise new text appears explaining why their answer was incorrect.

In a mathematical text, both the question and the various answers will likely contain mathematical formulas. But recall that the interactive material is being written in html and inserted directly in the final document without processing. Are authors expected to write the mathematics in MathML? If they write in LaTeX, TeX4ht cannot convert the code to MathML because it doesn't touch the author's html blocks.

The happy answer is that, with MathJax, authors *can* directly write LaTeX math, even inside the verbatim `{html}` environment we've defined (because MathJax recognizes the math). There is one caveat: normally, inline math can be specified by either a pair of `$` signs or a `\(` and `\)` pair, but inside `{html}` and when using MathJax, `\(...\)` must be used (or extra MathJax configuration specified). `$...$` works fine with MathJax outside of our `{html}` environment.

Display math can be defined by a pair of `$$` signs or a `\[` and `\]` pair; both these forms work inside `{html}`, with MathJax and otherwise.

```
\ifx\HCode\undefined
\else
 \section{New Experiment}
 \begin{html}
   <p>This sentence has <b>bold</b>
   and <i>italic</i> text.</p>
   <p>Also math: \(y = \sqrt{x^2 + 1}\) and
   $$\int_0^\infty e^{-x^2} \ dx =
   {{\sqrt{\pi}} \over 2}$$</p>
\end{html}
\fi
```

Typeset and you will see the output below (right margin has been truncated).

10 New Experiment

This sentence has **bold** and *italic* text.

Also $y = \sqrt{x^2 + 1}$ and

$$\int_0^\infty e^{-x^2}\,dx = \frac{\sqrt{\pi}}{2}$$

But how is this possible, since source inside an "html pair" is inserted directly in the output without processing?

9 Calling TeX4ht

Originally TeX4ht output small pictures for inline and displayed mathematics. Eitan Gurari unexpectedly died in 2009, and TUG paid him the ultimate compliment by keeping his program alive. Now it is actively maintained by Michal Hoftich.

Due to new developments in MathML and MathJax, there are many ways to call TeX4ht when it is asked to typeset. Let us concentrate on the three most important methods.

Calling TeX4ht using the call

```
make4ht source.tex "mathml"
```

causes TeX4ht to insert MathML code for inline and display equations. This MathML is then rendered by the browser.

Calling TeX4ht using the call

```
make4ht source.tex "mathml,mathjax"
```

causes TeX4ht to insert MathML code for inline and display equations, but call MathJax to render the resulting code.

Calling TeX4ht using the call

```
make4ht source.tex "mathjax"
```

causes TeX4ht to insert LaTeX code for inline and display equations, and call MathJax to render the resulting code.

Note that MathJax can render both MathML and LaTeX code when it discovers equations in an html document.

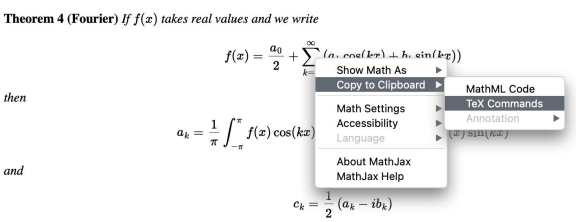
On my computer, mathematical rendering using the first method is not as clear as rendering with the other two methods. Integral signs are too small

and there are other minor flaws. The first and third methods understand LaTeX input for interactive content, but the second does not. These experiments suggest that the third method is the most desirable for interactive code.

My initial experiments did not go well with the third method. Inline equations were fine, but displayed equations were rendered with static images. Then one day I tried the alternate `\[` notation rather than `$$` and everything worked. I reported this to Michal, and the *very next day* he fixed TeX4ht so both notations are rendered with MathJax. (The LaTeX developers do recommend `\[...]`, by the way.) Please update your TeX Live distribution and typeset using the third method.

10 A MathJax perk

By now, perhaps you have typeset your own document with TeX4ht and MathJax. Select an equation and right click on it. A contextual menu opens offering to copy the equation to the clipboard as either "MathML" or "TeX Commands". Here's a picture:



Select "TeX Commands", copy, and paste somewhere else. You will obtain the LaTeX code for the equation. This code can be copied into any other LaTeX source document.

This remarkably useful feature comes from MathJax and is not available if you call TeX4ht using the first method. Moreover, the menu will offer MathML code, but not LaTeX code, if you call TeX4ht using the second method. But the third method of calling TeX4ht gives LaTeX code.

11 Installing documents on the server

Suppose you typeset a document named `Sample` with TeX4ht and produce `Sample.html`. How should this file be put on a server? The answer is tricky because `Sample.html` itself will not contain any images, so any needed image files must be provided separately. Moreover, TeX4ht generates a support file `Sample.css`, which is also required.

Thus it is convenient to put all illustrations in a folder, named (say) `Graphics`, and refer to these illustrations in the LaTeX source using the pattern `Graphics/plot1` (LaTeX will automatically look for usual image extensions, using whichever is found). Then the web server should contain `Sample.html`, `Sample.css`, and the `Graphics` folder.

12 Using the work of other people

Nothing in this document comes from me. When I discovered that TeX4ht produces completely acceptable web pages, I wondered if it would accept html code and send it unmodified to the html document. I asked Karl Berry, who thought it was possible and asked Michal Hoftich. Michal sent the method described here, but I didn't believe it was sufficiently general. So I started writing a sample document showing that the method could not display math, or handle YouTube videos, or accept Sage code. My sample simply proved the opposite.

I do not know a single MathML tag. I knew the American Mathematical Society recommended MathJax, but didn't know why. I don't understand how these technologies work.

Several years ago I downloaded Sage. But I didn't know that web pages could access a server so students who had never installed Sage could still read web pages with Sage content. When I realized that, I used Sage to graph simple functions. When it displayed a 3D graph and let me rotate it interactively, I almost fell off my chair.

It is strange that I had to learn these lessons over again, because LaTeX is a crucial tool for me and yet I have never read *The TeXbook*; TeX macros are crucial for my life and yet I don't know how to write a macro. We can do things in our lives because of the independent work of thousands of people.

13 PDF and HTML in mathematics

When I was a college sophomore, I took an abstract algebra course from W. Wistar Comfort. His lectures were crystal clear; you could copy the board, read the notes at home, and see every step in its proper logical order.

Later I took courses with a more rough and tumble atmosphere; the instructor seemed to be inventing right in front of our eyes, and sections of the board would be crossed out when a better idea presented itself.

Both lecture styles worked, showing the dual nature of mathematics. To me, pdf is for the final crystalline form of mathematics, and html is for the rough and tumble way it is invented. Euclid is pdf, but Legendre is html, and Euler is both.

◇ Richard Koch
koch (at) math dot uoregon dot edu
http://pages.uoregon.edu/koch/

A Refinements for TEX4ht

(Everything in this section came from Michal Hoftich, who we asked to review the above.)

A.1 \ifdefined\HCode

The main article uses

```
\ifx\HCode\undefined\else ... \fi
```

to insert material only when processing under TEX4ht. This is fine, and is the general form. But when only the html output needs the extra attention, it can be simplified to:

```
\ifdefined\HCode ... \fi
```

(By the way, `\ifdefined` is an ε-TEX primitive; LATEX has required ε-TEX, and some primitives beyond ε-TEX, for years now.)

A.2 \NewDocumentEnvironment{html}

The main article uses the `{html}` environment inside `\HCode` conditionals, so that only TEX4ht sees it. This is fine, but it is arguably nicer to define the `{html}` environment in all cases, and make it a no-op when being processed for pdf (or dvi, but we won't keep mentioning that).

Also, we may as well define an analogous environment for material that should only be processed in the pdf case.

This can most easily be done using the relatively recent (2020) `\NewDocumentEnvironment` command. The following two definitions in the preamble define an `{html}` environment to ignore its contents (since normally we are running LATEX, not TEX4ht), and the `{pdfenv}` environment to typeset its contents (for the same reason):

```
\documentclass{article}
\NewDocumentEnvironment{html}{+b}{}{}
\NewDocumentEnvironment{pdfenv}{}{}{}
```

Then, in a configuration file for TEX4ht (see next section), we reverse the definitions so that `{html}` is active and `{pdfenv}` is a no-op:

```
% (in a configuration file, see below)
\ScriptEnv{html}
 {\ifvmode\IgnorePar\fi\EndP\NoFonts\hfill\break}
 {\EndNoFonts}
\RenewDocumentEnvironment{pdfenv}{+b}{}{}
```

Then the environments can be used without any conditionals. As a side benefit, the environments can be nested. For example:

```
\begin{document}
...
\begin{html}
<p>This is output only in HTML, but can include
   LaTeX math: \( a=b^2 \).</p>
\end{html}

\begin{pdfenv}
Nested \LaTeX\ not in the HTML output.
\end{pdfenv}

\begin{html}
<p>Then we can have more HTML.</p>
\end{html}
```

A.2.1 \NewDocumentEnvironment explanations

You may be wondering what the `+b` means in the `\NewDocumentEnvironment` call. If you're not wondering, skip this section.

The environment name (e.g., `html`) is the first argument to `\NewDocumentEnvironment`. The second argument, with the `+b`, defines how arguments should be handled. The third and fourth arguments, empty for us, define the code which is run at the beginning and end of the environment, respectively.

The `b` argument specification says to pass the body of the environment as argument `#2` to the code blocks. (`#1` is for the optional argument, which we don't use.) The `+` specifier allows multiple paragraphs within the environment body.

Since we don't specify any code to run, nothing is done with the environment body, so it is effectively discarded. On the other hand, when the argument specification is empty, the environment body is processed normally.

Many powerful argument specifiers are available, and they can be used when defining either environments or commands. See the LATEX `usrguide3` document for details.

A.3 TEX4ht configuration files

TEX4ht supports configuration files, which are a convenient way to specify document-wide settings. The environment redefinitions shown above are one example. Here is another example, moving the Sage specifications to the html page header (via `@HEAD`):

```
\Configure{@HEAD}{%
   <script src="https://sagecell...\Hnewline
   ...
   % must escape the # character:\Hnewline
   inputLocation:  '\#mycell',\Hnewline
   ...
   </script>\Hnewline}
```

Because the configuration file is ultimately TEX code, it is necessary to escape `#` with a backslash, and explicitly insert newlines in the output with `\Hnewline`, as shown.

If the configuration file is saved as `conf4ht.cfg` (the name can be anything), the `make4ht` call becomes:

```
make4ht --config conf4ht.cfg source.tex "mathjax"
```