

Dirk Bollaerts & Céleste Paalvast

[Home](#) [Mathematics](#)

∇f ∇f ∇f ∇f ∇f

Practical notes on using MathJax code in a website

1. [Introduction](#)
2. [How to build a website using MathJax](#)
 - i. [Building a minimal website](#)
 - ii. [Website containing TeX code](#)
 - iii. [Testing the code](#)
 - iv. [Restrictions on the TeX language constructs](#)
 - v. [Unvalidated HTML code](#)
 - vi. [Start immediately with online experimenting on the web](#)
3. [How to build a website using MathJax without TeX by replacing TeX with MML](#)
 - i. [Capturing the MML code rendered by the server](#)
 - ii. [The contents of server generated HTML file](#)
 - iii. [The injected CSS code](#)
4. [Some history and terminology](#)
 - i. [SGML](#)
 - ii. [XML enters the field](#)
 - iii. [HTML](#)
 - iv. [MathML](#)
5. [Links](#)
6. [The downloadable ZIP file](#)
7. [Contact](#)

1. Introduction

These notes were written down when we worked on our first project using [MathJax](#). These notes are not meant to be a tutorial. They can be of use if one is interested in experiments.

We are a novice in using MathJax but are quite acquainted using TeX. There are many people who do not use TeX in their website but use TeX rather exclusively for making pdf files that are then made downloadable on their website. We do not follow this strategy for the following reason. There are visitors of a site who take the trouble or effort to do the download and take the time to see if the text in the pdf is exactly what they need for their purposes. Some visitors have not the patience for reading text material but really want to see without too much effort what is offered exactly in the site. This in itself can make it worthwhile to consider to give a kind of preview about what they can expect when looking further.

A second reason could be simply a matter of politeness or service to the visitor of the site. The visitor is informed as well as possible right away what he can expect when he downloads the pdf document.

A last reason can be that the technology to do so is now readily available. Most popular browsers like Firefox, Chrome and Edge now support MathML right away in their browsers. It is almost an open invitation to let authors and website developers use this new technology immediately in their publishing web code. The strict boundary between web technology and mathematical typesetting is now rapidly fading.

The purpose of MathJax is to develop a webpage in which we can include TeX code. We can write TeX code inline e.g. $a = b$ or displayed like $a^2 = b^2$. MathJax will then do what we are used to do in TeX. It will make nicely typeset mathematical text of them. In this case the output medium is the screen of a computer and the programming environment is HTML. This means in practice that one can read nicely typeset mathematics while browsing the web.

We stress here the fact that if one is authoring with MathJax, one is essentially both a [HTML](#) and [CSS](#) writer because we essentially are writing code for the browser. But we are not hindered when it comes to show math in our HTML texts in the sense that we can use TeX in our texts. It is not so difficult to become versed in HTML and CSS. We give hints on how to start studying it [below](#).

2. How to build a website using MathJax

Let us start building a minimal website. Before continuing, we mention that all files we make in this text are collected in a downloadable [zip file](#). We do not have to do copy and paste but it is also easy to work in that way. Also the complete text of these notes in this page are offered in pdf form in this zip file.

2.i. Building a minimal website

Let us take first a complete minimal skeleton of a website without any MathJax or TeX code in it.

We show now a skeleton file that we will use a lot later on. We have added a skeleton file (`MINIMALskeleton.html` in the [zip file](#)) that shows the minimum code to start a project.

Let us take first a complete minimal skeleton of a website. We show now a skeleton file that we will use a lot later on. We have added in the downloadable [zip file](#) a skeleton file (`MINIMALskeleton.html` in the zip file) that shows the minimum code to start a HTML project without for the time being mentioning MathJax and TeX. This code should be evident for anyone who has studied the basics of HTML.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<title>MathJax v3 with TeX input and HTML output</title>
</head>

<body>
<!-- Page content of our website -->

</body>
</html>
```

2.ii. Website containing TeX code

We will now build our first website with TeX code in it. We start from our `MINIMALskeleton.html` file. We show first the complete code of the site and will highlight the code **added** to the `MINIMALskeleton.html` file. More detailed explanations follow after showing this code. We remark that we have added this file also in the [zip file](#). We have called this file **skeleton.html**. This small file can be used to take as starting point for experiments undertaken by the reader. We stress the point that this file is able to run without problems. So we have in a sense already an up and running site. This is a preliminary goal for this subchapter. We can go on [experimenting online](#) with this file by simply adding some TeX code or HTML code to it. So it is not necessary at this moment to install or download anything. We can start working online right away . It is also quite interesting to see what is in this file.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<title>MathJax v3 with TeX input and HTML output</title>
<script>
    src="https://cdnjs.cloudflare.com/ajax/libs/babel-
polyfill/7.10.4/polyfill.min.js?features=es6"></script>
    MathJax = {
      tex: {
        inlineMath: [
          ['$', '$'],
          ["\\(", "\\)"]
        ],
        displayMath: [
          ['$$', '$$'],
          ["\\[", "\\]"]
        ]
      },
```

```
MathML: {
  extensions: ["content-mathml.js"]
},
options: {
  menuOptions: {
    settings: {
      assistiveMml: false, // true to enable assistive MathML
    }
  }
};
</script>
<script id="MathJax-script" async=""
src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-cthtml.js"></script>
</head>

<body>
  -->
  own code. -->
  eorem that says
  $$a^2 + b^2 = c^2$$
  are the lengths of the sides of a rectangular
  triangle and $c$ is the hypotenuse, then we have that
  $$
  c^2 = a^2 + b^2.
  $$
</body>
</html>
```

We see first that in this piece of code there is no TeX preamble because it is actually HTML code text. Remark on lines 41-46 that the HTML code is interspersed with inline TeX code

delimited between single dollar signs “\$” and displayed TeX code delimited between double dollar “\$\$” signs.

Let us take a look at some fragments of this code. We start with the lines 7-8.

```
<script>  
  src="https://cdnjs.cloudflare.com/ajax/libs/babel-  
polyfill/7.10.4/polyfill.min.js?features=es6"></script>
```

This is a script that take cares of JavaScript incompatibilities between different versions of JavaScript. It is of importance if the visitors of our website use old versions of JavaScript.

The second fragment is on line 32.

```
<script id="MathJax-script" async=""  
src="https://cdn.jsdelivr.net/npm/MathJax@3/es5/tex-cthtml.js"></script>
```

This piece of code takes care of the fact that all the TeX code in our page must be automatically changed in HTML code that can be understood by a browser. It is the task of MathJax to do exactly this. All the TeX code in the page will be interpreted and transformed into HTML code and it is this HTML code that is shown on the client machine. This transformation is necessary because a browser understands only HTML. What happens is that the machine of our client asks to another computer, the so called server, to translate the TeX code fragments into HTML. More on this client/server model follows.

The client is in this case the computer of the visitor or reader of our site. The browser of his computer reads this script command and asks the server to translate the TeX gibberish in understandable HTML code. The server is the machine in the cloud that is the keeper of the necessary JavaScript MathJax software to do the translation into HTML code. It is the server software that silently converts our TeX code into HTML code on the machine of the visitor of the site. This server software resides in the server of the URL `cdn.jsdelivr.net`. The above code is a request from the machine of our visitor to the server.

The code that is downloaded from the server contains the MathJax fonts, the necessary CSS code and all further code that is necessary to typeset our TeX code in the browser.

The string “es5” in the code is responsible for using the correct version of MathJax. That version of JavaScript is ECMAScript 2009.

The purpose of the code “@3” is to take care that the latest version of MathJax 3 is used.

The disadvantage of this way of working is that there can be a small delay in the translation from our TeX code to HTML code which is the only code well understood by browsers. This delay is quite small but it can be a disadvantage when the visitor is surprised to see that the loading of the website is not as fast as he expects it to be.

We see when looking at this code that one is essentially a HTML author writing HTML code but mixed and interspersed with pieces of TeX code either in inline style or in display style. If the reader is not very well versed in HTML, then he can start with something like an excellent [tutorial text](#) of Michael Gabriel. There are of course numerous other texts. We can advise a search in our favourite browser. For basics in CSS, one could advise [CSS basics](#). For the topic of selectors, one could advise [CSS Selectors](#).

We concluded from this code that the main body is just ordinary HTML. This HTML code is just interspersed with TeX inline and the display style code. These pieces of TeX code are the pieces of code that MathJax carefully slices out of the code. Then MathJax translates these pieces in MathML code. There follows a subsequent conversion from MathML to HTML. This transformed code which is now packaged as ordinary HTML code is send back together with all necessary math fonts to the browser of the client. His browser inserts it at exactly the same places where the TeX code was sliced out. There is actually much work done behind our back by the MathJax code and it is almost a miracle that it succeeds to do that in a fluent way.

Some explanations about the work of the CDN server mentioned at the end of the heading section on line 32. The name CDN is an abbreviation of [content delivery network](#). A content delivery network consists of a set of servers that are geographically placed in almost every part of the planet. When they get a message that work has to be done that is meant for them, then they do that work invisible for the web author and the visitor of the site. When that work is done, they send the result and output to the computer of the visitor. At this moment of writing, it is the `jsdelivr` network that does this service for MathJax. The work done by these servers cannot be underestimated. They process several terabytes of MathJax TeX code in a year.

We take now a look at the code fragment

```
MathJax = {
```

```

tex : {  inlineMath: [ ['$','$'], ["\\(", "\\)"] ],
        displayMath: [ ['$$', '$$'], ["\\[", "\\]"] ]},
MathML: {
extensions: ["content-mathml.js"]
},
options: {
  menuOptions: {
    settings: {
      assistiveMml: false    // true to enable assistive MathML
    }
  }
}
};

```

We first defined the delimiters of math in inline TeX and displayline TeX. We try to avoid that there are users who have to change their working habit by offering the use of two standard ways of working with math delimiters.

MathJax adds what is called assistive MML. It is not clear to us which purpose this assistive MML is serving. If we visit the example files of MathJax in github, we can right-click the math in the HTML file and we see a menu popping up. We do not understand the practical purpose of this menu. We also looked for this purpose in the docs but did not find it. So we always clear this assistive MML from our source files because it takes up **very much space** in the HTML files. It nearly doubles the size of these files and we try to avoid that at all cost. If one takes a good look at the `skeleton.html` file, we see that there is a tag called `mjx-assistive-mml`. We show here the tag with the code between it. This code is for our purposes unnecessary and we will delete it from our files to make the transmission of the HTML to be as fast as possible. If we think that this code is something that adds content to our website then we can leave it as it is. It seems in this example not to take much space but experiments in real life examples show definitely otherwise. So we decide to delete it. If the reader finds this assistive MML useful, he can include it of course by writing `assistiveMml: true`.

2.iii. Testing the code

It is not too difficult to experiment with our MathJax HTML file. The reader has probably already developed his own way of HTML authoring. It can be helpful for novice users to advise

the following. A way of working is writing the HTML code with a text editor or a dedicated HTML editor. We highly advise to save the file with the file extension “.htm” of “.html”. We open our favourite browser. Instead of opening a website in the browser, we go to the file menu of the browser and choose for “open file”. We remember that a file on our file system has also a valid URL and the browser is by definition able to browse a URL and so it can read all our files on our system. We open then the HTML file we just saved. We should be able to see now the website in its full glory. It is of course necessary that we are connected to the internet or the CDN cannot do its work. An alternative way of working could be to double-click on the file in our file system. There is really nothing more to be done. In a next stage we can look for facilities in our favourite text editor to launch the browser from within the editor with a combo key click. Almost every serious editor has a facility of this kind. We can preview in this way the result without too much overhead.

We can do a very fast check with our test file. We copy with copy/paste the code out of this text or [download](#) the test file `skeleton.html`. We save it somewhere on our disk. We open in our browser this file using the “open file” facility in the “file” menu of the browser. This action is browser dependent. We use the procedure for opening HTML files in the filesystem provided by our browser. When we open the file and everything goes well, then we should be able to see on our screen something like the following figure.

All children on this planet learn the Pythagorean theorem that says that if a , b and c are the lengths of the sides of a rectangular triangle and c is the hypotenuse, we have that

$$c^2 = a^2 + b^2.$$

2.iv. Restrictions on the TeX language constructs

MathJax understands many TeX expressions but not all. It takes some time getting used to this fact. MathJax chokes sometimes on TeX expressions that are defined in many LaTeX packages that are written in the so called style files. If we are used to develop TeX/HTML files then one can almost feel what is accepted and what not. We have not encountered any problem at all. Most typesetting of pure mathematics always works. Style files that are aimed to do things like restyle the page and restyling the headings and section commands will probably not work, because they are not necessary as this work is done in HTML. As a rule of thumb: we can take out of TeX source all commands that are related to typesetting mathematical expressions. We can certainly copy all of our mathematical TeX expressions in our HTML file. The AMS- TeX code is also supported. Special style files do not work in general.

If one heavily relies on special style files, then one has to experiment to see what works and what not works. Very specialised style files will probably not work.

We think that it helps to think HTML-like for the logical structure of the HTML page but to think TeX-like for the mathematical formulas we use in the HTML page. So we think about `<article>`, `<section>`, `<h1>`, `<h2>` and `` obviously in HTML terms. We think however TeX-like for our formulas, like `\begin{array}`, `\int` and `a = b`.

People who rely heavily on the classical and traditional tabular environment of LaTeX will have problems though. This phenomenon is probably caused by the fact that the HTML for constructing tables is very well and extensively developed and is powerful enough to satisfy almost everybody's wishes. There is definitely not a real additional value for the LaTeX `\begin{tabular}` command.

There is a very fine [help page](#) containing information about which TeX code one can use written by Carole Burns. There is also a [pdf](#) file in it that can be downloaded. It lists meticulously every TeX command that can be used by MathJax and gives a short description and example of every one of them. This documentation is particularly written for the 2.x versions of MathJax and we are now using at the moment of this writing MathJax version 3.2. We personally do not feel that it is outdated. We never encountered any trouble when using it and we used it frequently.

If we want to have an idea about the capabilities of MathJax, we can take a look at the [demos](#) of the MathJax project. It is especially interesting to see a kind of [TeX-MathJax torturing page](#).

If one wants a *very short* but quite useful introduction on which TeX commands we can use, then this [StackExchange page](#) with the title “**MathJax basic tutorial and quick reference**” is a good place to start.

2.v. Unvalidated HTML code

HTML code was in the first years of its existence treated by the browser as if every source code with correct or erroneous grammar of HTML was allowed. A HTML writer could do nothing wrong with his syntax and the browser forgave almost every error and tried to make the browsed code readable in one way or another. This attitude of the browsers made HTML without any question very popular. Everybody could write code and it was surprising that the browsers could do such a good job even when confronted with faulty code. It goes without saying that if one wrote erroneous HTML, then it was not guaranteed that what came out of the browser was what the HTML author intended. But the point was that some result came always

on the screen without a warning. There was never a strong policy of abruptly stopping the code interpretation caused by a syntax error.

It is in this tradition that HTML still continuously and silently passes blatant syntax errors. The author of the code is solely and without any help or assistance responsible for delivering serious and error-free code to ensure that what the browser made of the code is really the author's intention. Another unwanted danger of erroneous code is that it turns out alright in our brand of browser but that it ends up wrongly in another brand of browser. Still worse is a version error that lies dormant waiting for a future update of HTML to suddenly wake up and avenging itself.

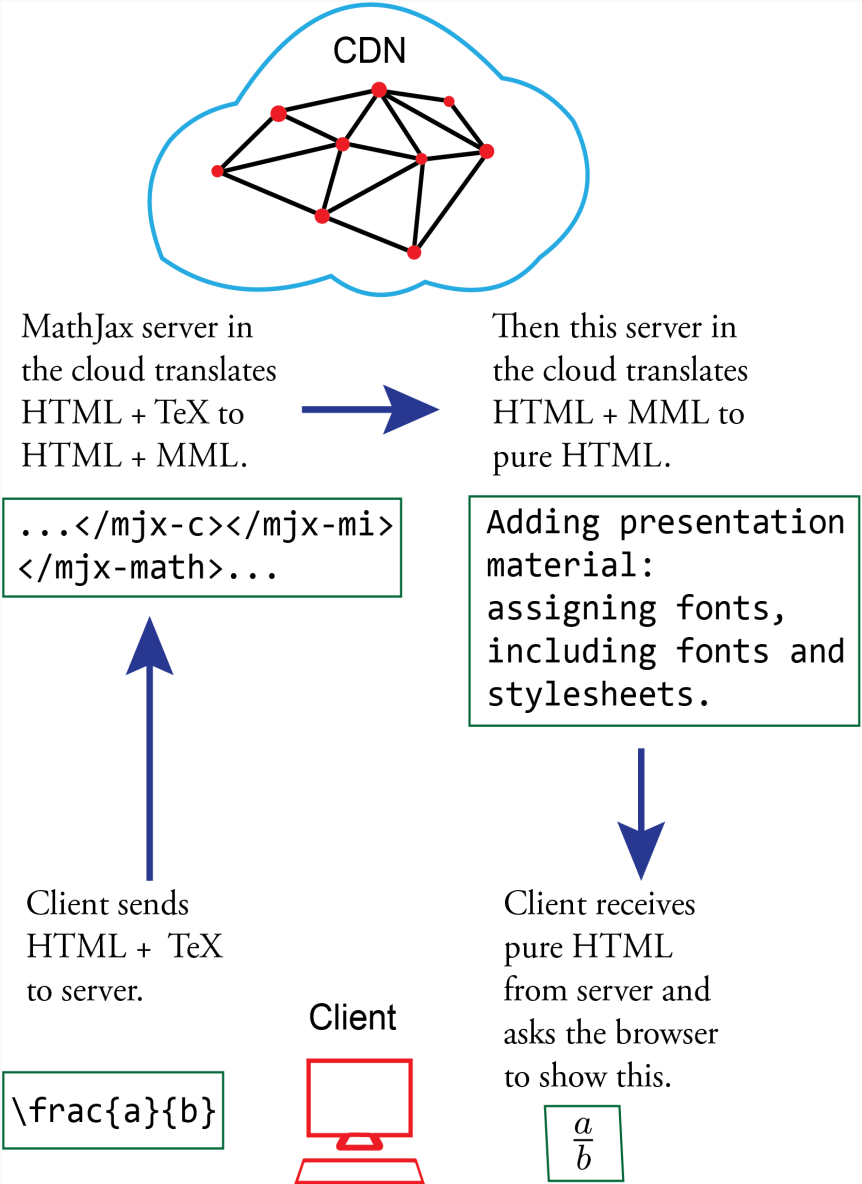
We have to realise that we work with one additional layer of complexity caused by TeX on top of the HTML code. This makes the endeavour even more error prone. We would advise to first check the TeX code thoroughly in a TeX environment. In this way one can be sure that e.g. all braces are alright and the spelling of the TeX expressions are valid. The JavaScript interpreter of MathJax is also very forgiving but there is nothing worse then a JavaScript that silently and mysteriously comes to a halt somewhere in the middle of a thousand lines HTML code text. It is for this reason that we advise the MathJax user to run their final code also through a HTML code validating engine before even starting to check the code with and in the browser. We can alas testimony also from our own experience that this can save countless hours of time.

There is still another problem that we have to cope with when we do not validate. It can be the case that we try to apply code that is officially part of HTML and approved by the W3C. But if one of the major brands has still not implemented it, then we run into problems unless we know ourselves which feature is still not supported by which brand. And it is highly unlikely that we keep ourselves busy with gathering information about that. So it is probably useful to ask this question to the validators.

We could advise three validating machines. There are doubtlessly many other good validators.

- [W3 validator](#). This validator must be used by uploading the code in the cloud.
- [CSS HTML Validator](#). There is a free version of this validator. This validator has to be installed on our computer and can be run on our computer.
- It is also possible to work with the well known [HTML tidy](#) program. It also cleans and beautifies our code if we want that, but more importantly gives interesting error warnings.

Let us end this chapter with an infographic that shows the way MathJax operates.



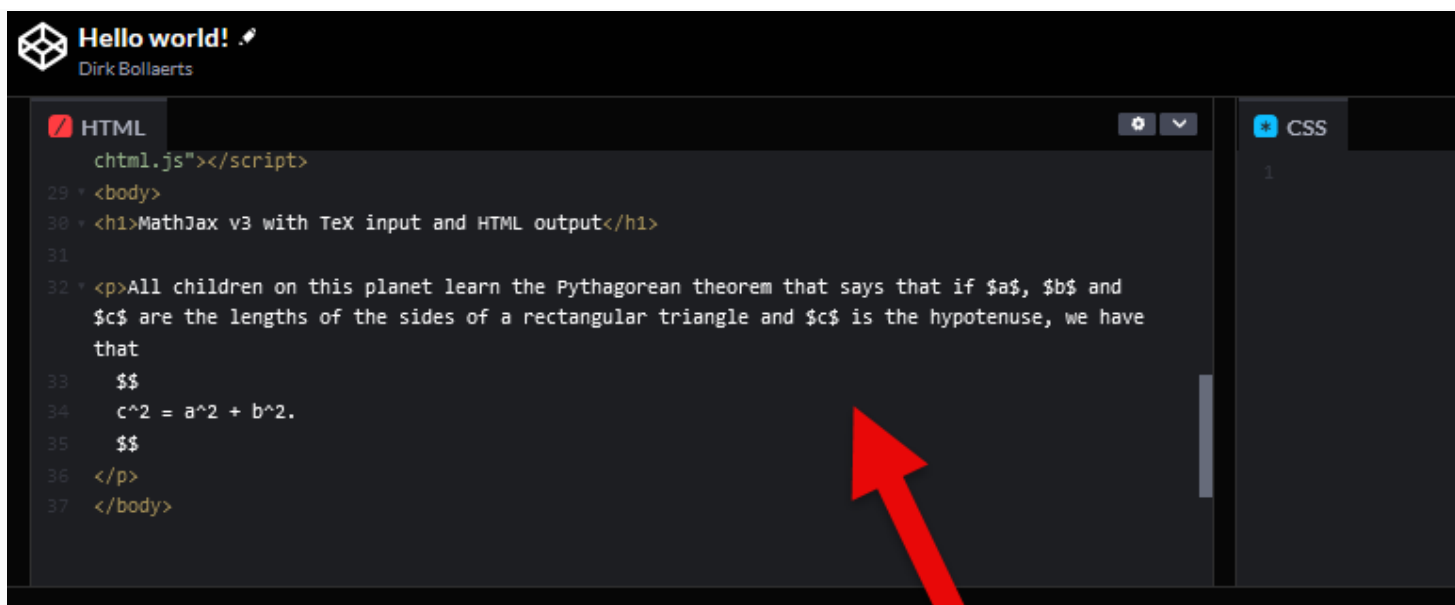
Translation Scheme TeX to MML. We send first our HTML code mixed with TeX code to our clients, i.e. visitors of our website. Then the code is send from the machine of the client to the server in the cloud. This server translates the code first to MML (more about [MML](#) later) and adds style elements and math fonts and sends the result back to the browser in pure HTML form.

Suppose now that we want to do all translations made by the CDN server ourselves and that we will try to send the code ourselves in HTML code to the browser of the client without interference of the cloud server. Then there is more work for us to do, but we have everything in our own hands. The time delay caused by the server can then be shortened. This is the target of the next chapter. The readers who are only interested in a modus operandi as described before that works well and is efficient can skip the next section and can optionally continue with [history](#) or [links](#) or the [download section](#). We show in the following section of this chapter a way to very quickly experiment with MathJax on the popular and very good CodePen website. We use for this purpose the skeleton file we have introduced before in this text. We do not forget to

mention that there is a downloadable [zip file](#) with the text of this complete webpage in pdf format and which contains all the code files that we talked about in this webpage in UTF-8 format.

2.vi. Experimenting online with MathJax code.

If the reader wants to experiment immediately with coding in MathJax, then he can do so online. He has to follow the following procedure. He has to click on the huge green button below after the screenshot that is at the end of this section. When that is done he will be in the website of CodePen. This is a site with facilities for previewing and developing web code. We click then on the yellow button with the text **“Push this button to have a new Edit Window and adapt the skeleton file to your wishes and experiment with the code here on CodePen!”** and we get a second CodePen window again divided in three black input windows in three columns shown in a black background above and an output window that is shown in white color. This is our editor. We use the black editor window at the top left and scroll a little bit to the end of the code until we see the tags `<body>` and `</body>`. Some text is already there which we wrote for our **skeleton file**. We can add some text and some TeX code at leisure. Experiment as you want. We cannot do any harm by experimenting. Add e.g. code like **We see now that $c^{2 \setminus, i + 1} = \frac{d}{e}.$** Always have patience and wait a little bit until the screen is refreshed and the result of your coding appears in the white output window below. MathJax must work now **very hard** behind the scenes because we are in **live preview mode** and that takes necessarily some time.



MathJax v3 with TeX input and HTML output

Editor Window

All children on this planet learn the Pythagorean theorem that says that if a , b and c are the lengths of the sides of a rectangular triangle and c is the hypotenuse, v

Output Window

A view on the working space of a typical CodePen window. We see at the top left the editor window with black background and at the bottom the preview window with the white background.

Click this green link button to start experimenting with publishing TeX by MathJax seen in your browser on your computer screen.

3. How to build a website using MathJAX without TeX by replacing TeX with MML

If one would like to make a website **without TeX** but including MML, then this is also possible. It is somewhat more complicated then the previous way of working, but it is certainly not very difficult.

Our strategy is as follows

1. We capture the code that is produced by the server. This can be skipped if we know how to

capture server generated code.

2. The server injects a lot of CSS code internally right into the preamble of our HTML file. For reasons of pure elegance, we make this CSS code external. This CSS code is quite large and we do not like to work with a bloated HTML file. If the reader is not interested into doing this, he can skip this. It is not necessary to do that. It is only a matter of coding style.
3. The server does a transformation first into mixed HTML/MML code. This code is then transformed by JavaScript residing in the server first into MML and in a further stage pure HTML code. This is code that a browser understands. We will now intercept the phase of making MML on our own developer's machine.
4. Then we send this MML code to our clients and the server has then to take care **only** of the translation of MML to HTML.

Our goal is to generate the HTML/MML translation on **our own developing machine** and send this code to the client. This speeds up the waiting time on the machine of our client because the server in the cloud can in that case skip the translation from TeX to MML.

We have the following task to accomplish. We have to find out how we can capture the MML code that is produced by MathJax.

We found out that it is very difficult to do that by using the config file of MathJax. There should be ways to do that and maybe the config file will be easier to manipulate in future versions. We found an elegant way to work around the config files when we looked at the sample files.

3.i. Capturing the MML code rendered by the server

We explain in the following a way to capture server generated code. It is certainly possible to take a look at the mixed HTML-MML code generated by the server. There is a well known and much used method to get access to the server source code. Because many users do not use this very often, we try to make this method clear and transparent. We will describe the method in the following sections. We do this only for the Firefox browser. If we are already acquainted with capturing server generated code, we can skip this section entirely.

We explain this now for the **Firefox browser**. It is as far as we know not possible to do that in Chrome.

It is necessary to add an important and excellent extension written by Chris Pederick for the Firefox browser. We [download](#) it. We choose the Firefox version and install it in our Firefox browser. Then we **execute** the HTML file with our TeX code in it like we described in the

preceding example. We can use for testing purposes the `skeleton.html`. This is the same file as the `skeleton.html` file of the previous chapter. We have written the following line for invoking the code on the server.

We open now and execute with Firefox our HTML file which is on our disk. We see now our TeX file as interpreted by MathJax nicely typeset on our screen. If everything works, Firefox shows our mathematics nicely rendered on its screen. We do not forget that our machine is supposed to be online. Otherwise the CDN server cannot do its work. Now select everything in our Firefox browser with CTRL -A. Then right-click somewhere on the screen and a menu shows up and we choose `View source code`. Now we see a file in UTF-8. We copy all the code in this file to our favourite text editor in UTF-8 text mode and save it somewhere in our work folder with a suitable name like e.g. `SkeletonMMLServerSource.html` or whatever else we like. This file is also available in the downloadable [zip file](#). We will work with the code in this file later on.

We do not know how to capture the server generated source code with the **Chrome browser**. We thought at first that it was easy by using the Inspector of Chrome. It is certainly possible to take a look at this source code in Chrome, but there is no easy way to copy that code entirely with a few keystrokes in our text editor. If we want to do that, we must copy every Element in the DOM which is no practical procedure. We searched the internet and found many discussions about this issue, but there was nothing useful for our purposes. The Pederick extension for the Chrome browser does not seem to help also. To the best of our knowledge there seems to be nothing useful for our purposes in Chrome.

3.ii. The contents of server generated HTML file

We have at this moment the following situation. We have developed a HTML file, `skeleton.html`, with TeX code in it. We kept this at our developers machine and have opened this file. If everything went well, we can see the result in our browser with beautiful typeset TeX in it. What is shown now on our screen is produced by the MathJax CDN server. This server has in the background transformed our source file in an intermediate mixed HTML-MML file and afterwards has translated that result in pure HTML. We capture now this generated code using the technique which we have just described.

The contents of this file are in our [downloadable zip file](#) in the file `SkeletonMMLServerSource.html`.

It is once again advised to first validate and check the TeX code in our favourite TeX engine and

the HTML code to see that everything is as we expected. If not, we change the HTML code and the TeX code until we are pleased with the result and everything is validated. Otherwise we have to repeat the complete procedure we describe now. We can suppose now that everything went well syntactically. After having done this we can let our machine carry the burden of the transformation of the code into MML and not the CDN server. We see that the server has created a lot of internal CSS in the server generated code. We will continue now with **cleaning the CSS code**.

3.iii. The injected CSS code

The CDN server injects a lot of CSS in the HTML file. We have observed that the injected internal styling CSS code is exactly the same and is constant regardless of the source code. There are three sections of CSS-style code. We will send this CSS code ourselves to the client. We include this CSS in the code of our website on the server of our ISP. Because we consider it a bad habit to put all CSS style code in the HTML file, this style of coding is called in technical terms “internal CSS”, we have separated this injected code in two files which we called `MathJaxStylesheet1.css` and `MathJaxStylesheet2.css`. Remark that one CSS section has an identity attribute. We include that section separately from the two others. We will link these CSS files externally to our website, this style of coding is called in technical terms “external CSS”. These files are also included in the [downloadable zip file](#) that we have created. We can read more about these two CSS files in the next two paragraphs.

We could leave this CSS code on its original place in the preamble and send it to the user in the way the MathJax server does it. It is just a matter of style. But including CSS in an external file is generally considered a better style. The remaining HTML file is then also much easier to handle in our text editor. It is certainly advisable to make the style files external if we want to edit the mixed HTML/MML extensively.

Let us take a look at the included CSS code in our HTML file.

```
<style type="text/css">.CtxtMenu_InfoClose { top:.2em; right:.2em;}
...
// It runs from line 32
// to line 613.
...
mjax-c.mjax-c2E::before {
```

```
padding: 0.12em 0.278em 0 0;
content: ".";
}
</style></head>
```

We repeat that we can opt for leaving these CSS files as internal code in our HTML file. Otherwise we can do the following. We can remove this CSS code ourselves. It is not necessary to do that ourselves because the two CSS files are already included in the [zip file](#). To be exactly clear what the first style file will be, we show the code. It is between

```
<style type="text/css">.CtxtMenu_InfoClose { top:.2em; right:.2em;}
/* Line 32 ... Line 70 */
.CtxtMenu_Menu .CtxtMenu_MenuClose { top:-10px; left:-10px}
</style>
```

We do not forget to delete the tags `</style><style type="text/css">` on the fifth line and tenth line of this block of code. We put all this code in `MathJaxStylesheet1.css`.

After we have erased this code, we erase all the code in the last style block which contains the attribute `id="MJX-CHTML-styles"` We put this erased code in the file `MathJaxStylesheet2.css`.

We have put all this CSS code also in the [zip file](#). We have highlighted in the following code the two lines we added to link to the external `MathJaxStylesheet1.css` and `MathJaxStylesheet2.css`. It is possible that we have to change the relative position of our CSS files in the links code of our site. This depends on the file structure of the code in our website. A warning is here on its place. The most frustrating error in HTML authoring is without any doubt bad linking URL's. We have to be extremely careful when writing out the addresses of the files which we link. We consult our favourite validator before testing the code.

This `MathJaxStylesheet1.css` is of course also in the [zip file](#) that can be downloaded.

We have made the CSS code external and our definitive HTML file containing only MML takes its form. We will now set out to describe our new preamble of this definitive file. We leave out the CSS but have to include the external CSS code in our new preamble.

We have now finished the translation from TeX to MML on our own machine. So we could ask

the server to do now the following work described in this header.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<title>MathJax v3 with TeX input and HTML output</title>

<!-- This script causes older JavaScript engines
      to behave as more recent JavaScript engines do. -->
<script> src="https://cdnjs.cloudflare.com/ajax/libs/babel-
polyfill/7.10.4/polyfill.min.js?features=es6"> </script>
<!-- The following script tells MathJax to use the old fashioned
"$" signs as valid for delimiters of inline math in our TeX.
We ask MathJax to also accept "\(" and "\)" as valid delimiters
for inline Math in TeX. -->
<script>
  MathJax = {
    tex: {inlineMath: [['$', '$'], ["\\(", "\\)"]],
          displayMath: [['$$', '$$'], ["\\[", "\\]"]]};
  }
</script>
<!-- The following script tells MathJax to run the code
of the website first trough the cdn network server called
"jsdelivr". This network server will translate our TeX
code in HTML that is understood by the HTML browser of the
visitor of our sites. The necessary math fonts will be
temporarily added to the browser of the visitor. -->
<script type="text/javascript" id="MathJax-script" async=""
      src="https://cdn.jsdelivr.net/npm/MathJax@3/es5/mml-stylesheet" /> <!-- add
this line -->
<link id="MJX-CHTML-styles" href="../CSS/MathJaxStylesheet2.css"
rel="stylesheet" /> <!-- add this line -->
```

</head>

Remark at line 27 that the server has to do the translation from `mml -> chtml` only. It does **not** have to translate `tex -> chtml`. That is the reason why we changed `tex-chtml.js` to `mml-chtml.js`. This could mean a significant improvement in the amount of time before the user sees his math appear on his screen.

This file is also in the [downloadable zip](#) file under the name `SkeletonMMLServerSourceDefinitive.html`.

4. Some history and terminology

4.i. SGML

One could wonder why we need another technology then the TeX technology that is widely used and satisfies most needs in mathematical typography. The reason for that is the following. The time origin for TeX is an era in which tagging languages became popular. These tagging languages were introduced in an era when we had large mainframes and the first laser printers were sold. These laser printers had a memory in which more then one font could be used. Besides the ubiquitous Courier font, one usually was offered also a serif font in a regular, a bold and an italic version. Publishing and printing with the help of the computer became possible. It was in this era around 1978 that [TeX](#) has its origins. The tagging languages in those days were all presentation languages. This means that the tags were at the same time indicating the style in which the content was to be typeset and the content structure. They were mixed and there was not a strict guiding line between the two of them.

At around the same time researchers at IBM were working on what was later called [SGML](#), an abbreviation of Standard General Markup Language. The first originating ideas about SGML were found in the sixties by Charles Goldfarb, Edward Mosher, and Raymond Lorie. The intended goal of [SGML](#) was creating a language that could be used to transport database information to other computers that were not necessarily identical computers or computers that did not have the same operating system. Those were times when a file on one computer could not be read on another computer. This is nowadays incomprehensible but in those days it was a huge problem. The first fundamental idea of SGML is that the most common technology that was shared by different computers and operating systems is the flat text file. This is text

made by a text editor, not to be confused with a word processor. So a database should be written in a text file. The second fundamental idea is that one should tag the different components of a database with preferably an indication of their meaning. Let us try to give a simple example. Suppose we have a database of names consisting of a first name and a last name, such as John Doe. Then we could represent the data of a database of three names as

```
<firstname>John</firstname><lastname>Doe</lastname>
<firstname>Abraham</firstname><lastname>Lincoln</lastname>
<firstname>George</firstname><lastname>Washington</lastname>
```

The computer receiving this information can then import with the help of a simple program the received information in its own database because it is evidently supposed to be able to read this flat text file. The transport of information should of course be send together with information about the meaning of `FirstName` and `LastName`. These tags do not have to be as clear as those we have chosen. In fact these tags are completely arbitrary and could be chosen at leisure. One can immediately see two important facts. The way in which information can be transported is very versatile. One can for example change the tags but one could also change the order in which the tagging occurs. The second important fact is that sending information is quite verbose. This verbosity is at the same time a blessing and a curse. It guarantees an enormous flexibility and almost limitless generality and universality but at the same time the volume of information to be send is quite large. One can remark that this principle can be immediately be applied to all kinds of information transport. It can be generally applied to information as long as the information can be mapped into ordinary language.

We see here a first advantage. There is an almost automatic distinction between content and style. The receiver of the database can choose to present the `FirstName` in blue colour, a Consolas font in bold at 14pt, followed by an empty space of 20pt. We see that it is easy to separate the content and the style. Later on, there came a standard about sending also a style file with the database that can be applied by the receiver of the database file. The receiver can do with this style file what he wants.

The SGML standard also stipulated among numerous other topics that the tagged file to be send had also to be written in a treelike recursive structure. This made it possible to read the file with algorithms that are very fast. Searching the database was then very fast.

4.ii. XML enters the field

One realised that the SGML language was too large to be handled with flexibility. So there was a sublanguage [XML](#) created in the nineties of the previous century. XML is an abbreviation of eXtensible Markup Language. Another reason for the need of a successor of SGML was to create a language that is extendible. This was deemed necessary because HTML became very popular around the same time. HTML was another dialect of SGML but was by definition not extendible. This SGML sublanguage is very popular and is widely used. XML has become a standard.

It is important to remark that when applying XML to do data transport, one has to write a syntax definition file for that particular data. This makes XML a general application. It can be adapted to any kind of data transport. So it can be used for any information exchange, including documents, databases and all kind of information that can be coded in computer language. One calls this definition file a DTD, an abbreviation for Document Type Definition or in more modern terms a Schema.

4.iii. HTML

When [Tim Berners-Lee](#) was looking in the nineties of the previous century for a way to do efficient data transfer between different computers with browsing programs, he decided that the fundamental ideas of SGML were applicable. So he wrote a specification of the language called HTML, an abbreviation of HyperText Markup Language. He wanted to define a language that is highly similar to SGML. He started from the outset to make this HTML language more forgiving and less restricted than SGML and this is probably the reason why this HTML language did succeed. Everybody could learn it rather quickly and could use it and browsers were extremely forgiving for syntax errors and made their best to make bad HTML code readable in one way or another. All kinds of syntax mistakes were silently passed by and at least still something from the intention of the authors survived on the screen of HTML code written with errors. It can be remarked that a strict XML variant of HTML was also created and that is called XHTML. For the last version HTML 5, there is also a strict XHTML version. It has to be remarked that in HTML, the idea of a link that was already in embryo present in SGML was expanded greatly and a brilliant idea of [Vannevar Bush](#) was realised. One could navigate from any section in a HTML document to any section of the same or to any another HTML document. The idea of a universal library had found its first implementation. The idea of the universal library is quite old and a good point to start investigating this very interesting topic is [wikipedia](#).

4.iv. MathML

The [W3C](#) was formed in 1994. It wanted HTML to be expanded so that mathematics could be printed in browsers. In 1994, [Dave Raggett](#) submitted a proposal for HTML Math. The immediate inclusion of math tags into HTML failed because a lack of interest of the major browser builder companies.

Then the initiative was changed into making a way of representing mathematics by a pure XML application. This gave the birth of MathML (Mathematical Markup Language).

We do not start investigating the fine points of MathML here. There is a very good introduction in a site dedicated to [MathML](#). The people behind that website got the permission to publish large excerpts of the book [“The MathML Handbook”](#) by Pavi Sandhu.

Let us take a look at small examples of XML.

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <msqrt><mi>x</mi></msqrt>
  </mrow>
</math>
```

We try to explain what this code tells us.

1. It starts with `$`, that tells us that what follows until `$` is a mathematical expression. The `xmlns` tells us that we have to take the meaning of the following tags defined by the `www.w3.org/1998/Math/MathML`. It can be possibly the case that other people also created an XML application into which a tag as `<mrow>` is defined and we want to avoid double meaning.
2. The `mrow` element causes its child elements to be displayed in a horizontal row.
3. The `msqrt` element typesets a radical sign above its child element.
4. The `mi` element tells that its child is a mathematical symbol.
5. The symbol to be used is `x`.

Many more examples of MathML can be read in the aforementioned book.

It is important to mention that two styles of writing MathML are possible. The first style is presentational MathML. This style emphasises the two dimensional writing and tries to be a good presentation of the printed page. We want to make a nicely typeset page. The second style is content MathML. This style emphasises more the content then the presentation. This MathML is in the tradition of LISP and makes much use of prefix notation. One would e.g. prefer content markup when posting a formula for students on a forum to be copied and applied for calculations on the individual computers of the students.

There is a very strict demarcation line between style and content. Styling is done in almost the same way as in classic HTML. If one wants a fraction bar that is styled thicker then usual, then one uses CSS to indicate that. In this way there is a strict distinction between the abstract fraction bar and the thickness with which one would like to have this fraction bar printed.

5. Links

The following list of links is by no means exhaustive. We think that they can be a good start when one is interested in more background information.

1. The official MathJax website.

[Official MathJax website.](#)

2. MathJax previewer on Github. Can be used to test, if our TeX code is compatible with MathJax.

[MathJax previewer.](#)

3. More documentation on fonts if we are not pleased with the used font.

[More documentation on fonts.](#)

4. MathJax demo video

[MathJax demo video.](#)

5. Making MathJax equations responsive on the web.

[Making MathJax equations responsive.](#)

6. Putting mathematics on the Web with MathJax. From the ww3 consortium.

[Putting mathematics on the Web with MathJax.](#)

7. MathJax font documentation.

[MathJax font documentation.](#)

8. Previewing the MathJax of TeX constructs if we are not sure what the output in our site will be.

[Previewing MathJax TeX constructs.](#)

9. Another way to test if your TeX code is acceptable for MathJax.
[Test your TeX code for MathJax.](#)
10. All kinds of MathJax videos.
[All kinds of MathJax videos.](#)
11. Editor with which TeX code can be used in MathJax from the university of Adelaide.
[Which TeX code in MathJax.](#)
12. How to extract MathJax code from the webserver. With comments from Davide Cervone, the lead developer of MathJax.
[How to extract mathjax code from the web page.](#)
13. A discussion about software solutions to render Math on screen. Contains interesting URL's for software similar to MathJax.
[MathJax render math on the web on all browsers.](#)
14. A conversion program to convert from Microsoft Word produced MML to LaTeX.
[MML to LaTeX.](#)
15. It is interesting to know that CodePen also supports MathJax.
[CodePen.](#)
16. Kindle book of Ryan Hodson on MathML. Ry's MathML Tutorial.
[Ry's MathML Tutorial.](#)
17. Kindle book of Stephen Bucaro on MathML. : MathML Made Easy With Examples
[MathML Made Examples by Stephen Bucaro.](#)
18. Book on XML.
[XML.](#)
19. History of the Internet on Wikipedia.
[History of the Internet.](#)
20. On a critical view upon MathML, one can consult the following article on the site of Peter Krautzberger.
[MathML is a failed web standard.](#)

6. The downloadable ZIP file

We can download the zip file. It contains all the files mentioned in the text including a pdf from this webpage.

[Download the zip file.](#) Here follows a list of the contents of this file.

- i. MINIMALskeleton.html
- ii. Skeleton.html
- iii. SkeletonHTMLServerSource.html

- iv. `SkeletonMMLServerSourceDefinitive.html`
- v. `MathJaxStylesheet1.css`
- vi. `MathJaxStylesheet2.css`
- vii. A pdf file of this webpage called `MathJaxInstallationNotes.pdf`

7. Contact the author

The author can be contacted by using the address “Dirk.Bollaerts AT protonmail.com”.



[Go to the top.](#)



[Go to the home root of this site.](#)