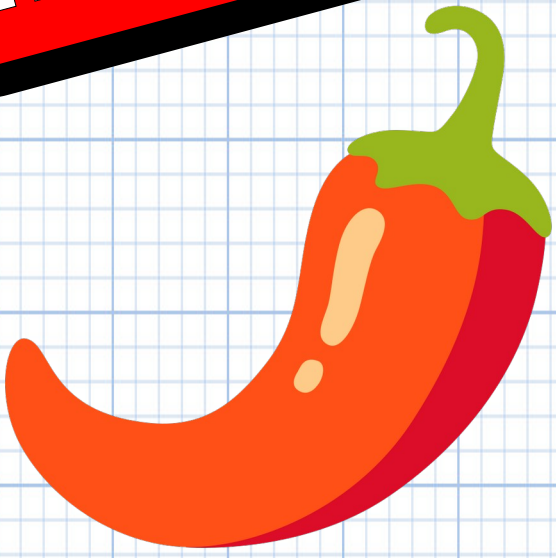
Learn HTML

the fun way



Programming basics for beginners

in easy-to-understand English!

– Sven Koerber-Abe –

Learn HTML the fun way – Programming basics for beginners

Version 1.7

For the newest version visit: http://sven.kir.jp/JS

This book was made using the following tools:

- Text editors: *Geany*, *Gedit*, *Kate* & *CotEditor*
- Asciidoc tools: *Asciidoc* & *Asciidoctor-pdf*
- PDF merge utility: *Preview*
- Front page chili & emoji graphics: *Noto Emoji*
- Graphics editor: *Gimp*
- Front & back page editor: *LibreOffice*
- Browser: *Firefox*

This book is released under a free license, meaning that you can copy & print it as many times as you like. If you work at a copy shop and are worried if copying or printing this book is legal: ***Yes, it is!*** *Please go on and tell anybody who might be interested about it.*

Learn HTML the fun way – Programming basics for beginners

Sven Koerber-Abe

Version 1.7

Contents

Acknowledgements

My biggest thanks to all the nice people at the *Aoyama Gakuin University Tokyo*, especially the *Faculty of Science and Engineering* as well as *Aoyama Standard* for supporting even my most unusual projects. You all rock! ;-)

Intro

What is this book series all about?

First things first: this book series called *Programming basics for beginners* is **not** for those who want to reach a professional level in web programming in as short a time as possible (those people better check out the more advanced resources in the chapter *Further reading* right away). The purpose of this book series is to show absolute programming beginners how easy it is and, most importantly, what fun it can be to create your own small webpages and programs that can be used on a variety of devices. Because with computers, tablets or smartphones you can do more than just play pre-made games or watch cat videos from the net.

Please don't get me wrong: games are a great pastime, and watching cat videos is an important basic human need - but it would be a shame to think that something like "programming" is far too difficult for oneself. With this book series, there is now a *very* simple and (sometimes) fun introduction to programming. Because writing your own little webpage and creating little practical apps is not that hard at all. And if you *really* have caught the programming fever at the end of this series, then with the programming basics presented here you already got the necessary basic knowledge to tackle more advanced programming stuff.

Who am I and why did I write this book?

My name is Sven Koerber-Abe, currently I work as an associate professor at the Faculty of Engineering of the Aoyama Gakuin University in Tokyo. My area of expertise here primarily is language education.

The reason why I wrote this book: while there are a lot of good and professional textbooks for HTML and JavaScript with the goal to produce advanced programmers, at the same time there is unfortunately also a glaring lack of freely licensed HTML and especially JavaScript textbooks for absolute programming beginners. And since I particularly wanted to have some textbooks that are written in simple English and therefore easy to understand even for non-native English language learners, I just wrote them myself.

Why HTML & JavaScript?

Depending on how many professional programmers you ask about how beginners should best learn programming, you will get just as many different answers. Many times you will hear that you have to learn programming with *that one* programming language (... most of the times, *that one* programming language happens to be the programming language with which the asked programmer earns his money...). And of course, all other programming languages are not suitable to really start learning to program.

So here comes an important fact, which only very few professional programmers publicly want to admit - but in secret will certainly affirm: actually, roughly all common programming languages are not that different. Honestly! The basic functionalities are quite similar, and that means that if you know one programming language well enough, you can easily switch to other programming languages.

So why does this book series teach *HTML* and *JavaScript* in particular? Surely there are other programming languages that run faster or are more modern? Well, there are several reasons for that:

- Web pages, programs and apps created with HTML and JavaScript run without any customizations on a wide variety of

brand-new as well as fairly old devices such as computers, tablets, or smartphones. Actually, any device that has a reasonably modern browser can run the webpages and apps presented in this book series.

- You don't need any special developer software to create HTML and JavaScript. A simple text editor is perfectly sufficient.
- Although HTML and especially JavaScript are powerful tools that can be used to create an incredible variety of applications, the basics are fairly easy to learn.

How this book teaches stuff

As already mentioned, this book series teaches just the basics of HTML and JavaScript programming without going too deep into complicated details.

This book, which is the very first book in the *Programming basics for beginners* series, is about using HTML to create attractive (and static) web pages that will look good in any browser. Among other things, projects tackle making HTML pages that look like pages from a daily newspaper, complete with headings, sub-headings and paragraphs. Another project involves embedding images and creating borders and shadows to make the web page look like a pamphlet.

In the second book of the series, called *Learn JavaScript the fun way*, we will bring movement into the HTML pages using JavaScript. Starting with quite simple projects like turning your own computer into a calculator, we'll move on to bigger and bigger projects like making a dog-years converter or a math training app. In the end, there is the final big project: creating a small science fiction game in which a rocket launch has to be performed.

All this may sound quite complicated and difficult - but don't worry! Everything is explained in a simple way for absolute

programming beginners seasoned with a small portion of fun.

What is this strange "Creative Commons" license all about?

This license means several things: first, it means that anyone can read this book for free and even share copies of it with anybody they know.

But it means much more: you don't have to take this book just "as it is". Everyone who wants to, may use the whole book or only parts of it to change it, expand or shorten it, combine it with other things and pass it on. You just may not charge money for it, you have to name the original authors and you have to put the resulting stuff under the same license. So this license doesn't just mean that you can legally use a book for free - it also means that knowledge itself is free and can and should be shared.

Why is there a chili pepper on the front page?

Because I like chili.

What tools do you need?

Computer

First of all, you need a computer. It doesn't matter if it runs *Windows*, *Linux* or *MacOS*. Your computer doesn't have to be the newest model, even older devices will do.

As for programming with HTML and JavaScript: the programs created with them will run on computers, but also on tablets or smartphones without any problems. And to be honest, **the HTML**

files in this book can also easlily be created with a tablet or smartphone. It is only important to mention that the **creation of JavaScript** in the next book is done on computers, because here you get an easy to understand feedback from the computer if something was programmed wrong.

Text editor

As for the necessary software, the nice thing about programming HTML and JavaScript is that you don't need big or expensive development tools. There certainly is a bunch of so called *Integrated Development Environments* (or *IDE* for short) and professional programmers are happily using them. But for a beginner to start with such software behemoths would be the same thing as using a F1 race car for learning how to drive.

You only need a so called *text editor*. It doesn't really matter which text editor you use, so please choose one that is easy for you to use. This book lists some popular text editors, all of which are free and Open Source, but there are plenty of others.

For a more detailed overview, Wikipedia has a comparison of the most popular text editors: https://en.wikipedia.org/wiki/Comparison_of_text_editors

On Windows, *Notepad++* is a popular choice (not to be confused with the software already included in Windows called "Notepad"). On Linux, there are *Gedit* or *Kate*, and on Mac, *CotEditor* would be a good choice. And finally, the text editor *Geany* is available on all three platforms.

- **Notepad++**: https://notepad-plus-plus.org/
- **Gedit**: https://wiki.gnome.org/Apps/Gedit
- **Kate**: https://kate-editor.org/

- **CotEditor**: https://coteditor.com/
- **Geany**: https://geany.org/

It is perhaps important to mention that you should **not** use a *Word Processor* or *Office Suite* for programming. Programming with these is possible under certain circumstances, but not really recommended for beginners, because word processors normally save files in formats that are not suitable for programming. Moreover, a text editor offers the advantage of so-called "syntax highlighting", where the respective parts of your own program are highlighted in different colors (more about this later in the book).

Browser

To see the web pages or use the apps presented here, only a browser is needed. Your device surely already has a browser installed, for example *Edge*, *Chrome* or *Safari*. With these browsers everything presented here works without problems. In this book the browser *Firefox* is used, for the simple reason that this one is free and Open Source. If you want to use it on your device as well, please feel free to install it (Firefox is available for a variety of devices and operating systems).

- **Firefox**: https://www.mozilla.org/en-US/firefox/new/

That is actually all that is needed for programming. So let's get started!

Code presentation in the book

Whenever this book describes programming code, it is displayed in a box like the one below. If you look closely, you may notice that some words of the code are displayed in different colors and some words even are displayed in a bold font. This coloring (and "bolding"?) of certain keywords is called "syntax highlighting":

```
<head>
    <meta charset="utf-8">
    <title>My first HTML file</title>
</head>
```

In your text editor you can turn syntax highlighting on or off somewhere in the menus or options. To be honest, you don't neccessarily need it if you don't want it and can just have all your code displayed in a single color and regular text. However, many programmers appreciate especially the color highlighting of code, as it makes it easier to see which parts of the code are, for example, certain keywords, comments, or text displayed later in the browser. It can also be an aid to avoiding errors when typing code: if you type in code and it is not colored as it normally would be, it may be a sign that you have written a small spelling error in that part of the code.

Each text editor has its own color palette specifying which parts of the code are displayed with which color, and some text editors even let the user choose from a list which color palettes to use. Maybe the colors in your text editor are a little different than the colors used here in the book, but please don't be bothered by that. The important thing is not the colors, but the code itself.

Where to save your files

Actually, you can store the programs and files that you write using this book anywhere you like all across your computer. But to be able to find your files when you need them in an easy way, a suggestion would be that you create in your home directory, e.g. in the folder `Documents` a folder named `JS` only for the programs of this book and store there everything described here in the book, and maybe your own programming projects, too. This way, there will be no frustration when you need that one file you spent so

long and effort writing, but now can't remember where you saved it on your PC. And if you *really* want to play it safe, it may be a good idea to sometimes save your programming folder on an external device like a USB flash drive as a kind of emergency backup (…because many computers somehow are able to sense when a human programmer needs the data stored on them most urgently and will have a hard disk crash at the exact moment when that programmer wants to access it).

Basic structure of HTML files

What is this "HTML" stuff anyway?

When you surf the Internet with your browser, the web pages are usually displayed in the form of HTML *(HyperText Markup Language)* files. Actually, if you only want to put some text on the net, you could just use a plain text file. (A surprising number of files, by the way, are put on the Internet as such simple text files, because they are easy to use and can be opened and read by a vast number of different kinds of devices). Basically, an HTML file is nothing more than a text file that contains other information besides the actual text, such as what background color to use, where in the text to put which photo, and so on.

So with HTML, simply said, "text is displayed nicely". Such HTML pages are usually only static, i.e. they do not react to user input or otherwise change during their use. For this purpose JavaScript, for example, is used within a HTML page: this programming language can be used to bring a wide variety of things to life, such as making calculations, moving game characters in a video game, etc.

If you really want to be extremely pedantic, web pages are rendered nicely using a combination of HTML and so called "CSS" *(Cascading Style Sheets)*.

First, let's start by using HTML to create a simple but nice looking web page. In the next book, we will then move on to enriching such HTML pages with JavaScript and thus creating small so-called "apps" (short form for *Application software*) from it.

Start your text editor and write a first line of text:

```
Hello World!
```

Save this file and name it `Hello.html`. Please make sure that the filename extension is `.html` and not `.txt` or something else.

You can now open this file in a browser like Firefox or any other browser of your choice: either you start your browser and open this file from there (use the menu item "Open file..." in your browser). Another way to open the HTML file is in the file manager of your computer like *File explorer* (on Windows), *GNOME files* or *Dolphin* (Linux), or *Finder* (Mac): there you right-click on the file, select "Open with..." and choose your browser. In your browser you will see something similar to **Figure 1**.

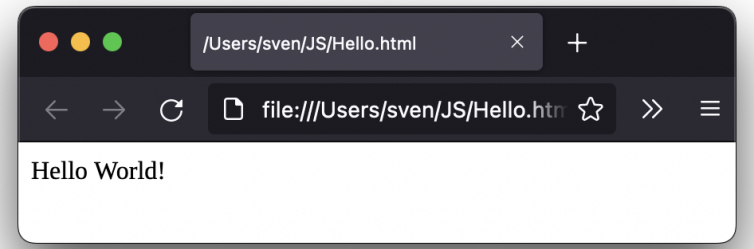


Figure 1. Your first HTML file `Hello.html` in the browser

If you want to open the HTML file you created in the text editor instead of the browser, it works in the same way: Either you select the menu item "Open file..." in the text editor and select your HTML file, or you right-click on the HTML file in your file manager and select your text editor under "Open with ...".

Ha! There it is, your first HTML file. Now you can copy this file and send it to other people, and they can open and view it in their browsers. Or you could upload this file to a web server and other people can view it on the Internet (this is exactly the way

webpages usually work).

Note that every computer and every browser may display the HTML file a bit differently: on one computer the font is somehow a bit bigger, or with another browser the font looks somehow different. This is quite normal, because not every computer and browser is the same as the other.

But to be completely honest with you, that file is actually not a *real* HTML file … it's just an ordinary text file that was simply given the filename extension `.html`. So let's turn our first fake HTML file into a real one!

The following line must always appear at the beginning of an HTML file:

```
<!DOCTYPE html>
```

This tells the browser that this is not an ordinary text file, but a *real* HTML file. Then follow two more so called *tags* to indicate where in the file the HTML code begins and where it ends:

```
<!DOCTYPE html>
<html>
</html>
```

Between the `<html>` and `</html>` tags you will write your HTML code. You could say that the `<html>` tag is the "opening tag", and `</html>` is the "closing tag". (Please note that closing tags always have that `/` character!) You will find this kind of opening and closing tags in other places in similar form when writing HTML. Always be careful with these tags that you don't forget the closing tag! Sometimes programmers write an opening tag, write more code, and then forget to add the closing tag at the end. An advice

would be that when you write an opening tag, always write the closing tag at the same time as well.

Within the two *html* tags there are two more tag-pairs, namely *head* tags and *body* tags:

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
    </body>
</html>
```

Between the opening and closing `<body>` tags you write the text you want to be displayed in the Browser. For example, here we would write our "Hello World!" text. *"Okay, but what are the* `<head>` *tags for?"* Between these tags you, for example, write some information that is used by Internet search engines or the title of your HTML page.

Our "Hello World!" example would be as follows:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>My first HTML file</title>
    </head>
    <body>
        Hello World!
    </body>
</html>
```

The line `<meta charset="utf-8">` in the head just tells the browser

what character encoding it should use. If you aren't using foreign languages that require rare special characters, you should always be fine with `"utf-8"` which covers most of the common languages used in the world.

The `title` will be displayed in the browser in the header bar or in the tab.

Maybe you are wondering why the different parts of the code are indented? Actually, you don't have to indent anything when writing HTML, you can start everything at the very left margin. However, indenting helps to quickly identify which things are parts and subparts of what tags. For example, here in the example, it's immediately obvious that the title is part of the head. This book uses 4 spaces as one indentation level in each of the codes. So in the example code, the head tags are indented with 4 spaces, the title is indented with 8 spaces, and so on. You can use as many spaces as you like if you too want to use indentation; an important advice would be to keep it constant and not use different numbers of spaces in your indentations in the same file. This could quickly become confusing. And please don't worry: the browser doesn't care if or how many spaces were used at the beginning of a line for indentation - it simply overlooks them and is only interested in the actual HTML code.

Just as it doesn't matter how many spaces you use *at the beginning* of your lines, it doesn't matter how many blank lines you use *between* your code lines. You could write one or more blank lines between your code lines and the result in the browser would always be the same:

```
<!DOCTYPE html>

<html>

    <head>
```

```
        <meta charset="utf-8">
        <title>My first HTML file</title>
    </head>

    <body>
        Hello World!
    </body>
</html>
```

Maybe with some space between the lines it's somehow easier to read for you? Why don't you try it yourself and insert some blank lines between your code lines in your example file, see the result and choose whatever fits you best.

This book will continue to use 4 spaces for indentation and relatively few blank lines in the code.

So now please write the code of the new example with your text editor and save this file as `Hello_2.html`. If you copied everything correctly and open this - *now real!* - HTML file in your browser, you should get something similar as in **Figure 2**.

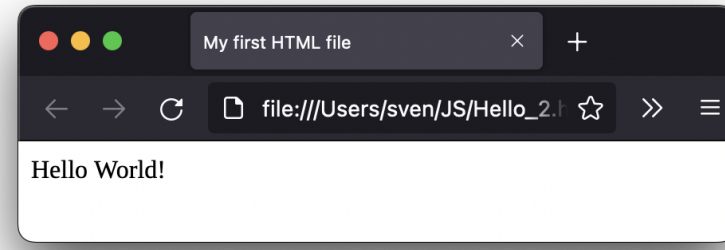


Figure 2. The (now real) HTML file `Hello_2.html` in the browser

"Well, this looks ... just the same as our first fake HTML file!?" Okay, for now there isn't much difference in the looks between a regular text file and our HTML file, but from now on we can use other HTML tags in our real file to make it look better, for example make the text bigger, choose colors and so on!

Empty HTML file

By the way, it is not necessary that you now learn by heart how the basic structure of an HTML file exactly looks. To make it easier for you, you can save this structure as an "empty HTML file" on your computer and when you want to write other HTML files in the future, you can always use this empty file as a template. The following would be such an example file named `empty.html`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>empty html</title>
    </head>
    <body>
    </body>
```

```
</html>
```

HTML formatting

Let's go ahead and beautify a regular text piece by piece with the various tools HTML offers. And as mentioned earlier, this book doesn't go through all the last details of HTML formatting, but only the most important and common things that you might need in your daily life.

Paragraphs & line breaks

A little surprise awaits the one who enters a text with several sentences, line breaks and blank lines. Try it yourself and enter the following HTML code into your text editor:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Linebreak example</title>
    </head>
    <body>
        This is the first sentence.

        This is the second sentence.

        This is the third sentence.
    </body>
</html>
```

Saved as an HTML file (let's name this file `Linebreak_Test1.html`) and displayed in the browser, it will look like **Figure 3**.

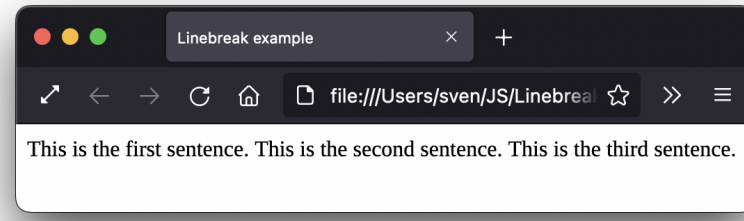


Figure 3. Regular text displayed in the file `Linebreak_Test1.html`

"Huh? In the HTML file every sentence is on its own line with empty lines in between, but in the browser, all sentences are on a single line?" Yup, that's exactly what had been described a few pages earlier: the browser doesn't care about spaces and blank lines at all, it will only display the "written" text. This means that you could write hundreds of blank lines between the text lines - the browser will only display the "written" text. It sometimes has its advantages, but what should you do if you want to write a sentence on a new line or have a blank line between certain sentences? Well, you'll have to tell the browser where it should insert line breaks, and there is actually more than one way to do that. One way is to use paragraphs. You simply write a tag in your text where each paragraph begins and where it ends. The browser then inserts line breaks between the paragraphs. The opening tag for a paragraph is `<p>`, and the closing tag is `</p>`. Let's make it so that in the previous example, the first two sentences belong to one paragraph, and the third sentence belongs to the next paragraph. The code would then look like this:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Linebreak example 2</title>
    </head>
```

```
    <body>
        <p>This is the first sentence.

        This is the second sentence.</p>

        <p>This is the third sentence.</p>
    </body>
</html>
```

Saved as `Linebreak_Test2.html`, it will look like **Figure 4** in the browser.

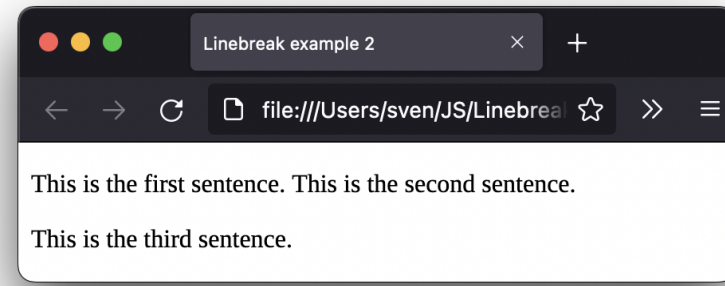


Figure 4. Two text paragraphs in `Linebreak_Test2.html`

If you view an HTML file in the browser, then change something in this HTML file with your text editor and save it, and then go back to the browser, you may not see the changes you just made: the browser first has to load the new version of the file. You can do this quite easily by clicking the *"reload"* button in the browser.

Another way to make a line break is to insert the line break tag `<br>`. Please note that there is no closing tag to `<br>`; you could say that `<br>` is a "stand alone tag". So inside the first paragraph in the

previous example you could insert
 after the first sentence to let the browser do a line break there:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Linebreak example 3</title>
    </head>
    <body>
        <p>This is the first sentence.
        <br>
        This is the second sentence.</p>

        <p>This is the third sentence.</p>
    </body>
</html>
```

The result will look like **Figure 5**.

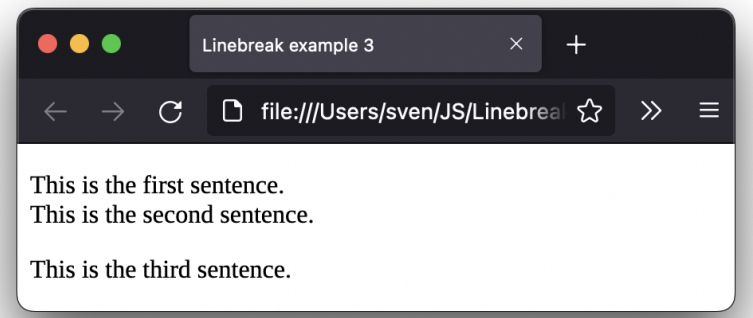


*Figure 5. A line break inside the first paragraph with
*

"Wait, there is no blank line between the first and the second sentence! There is a blank line between the second and the third

sentence, though." This is because the browser not only inserts a line break *at the end of a paragraph*, it also inserts a blank line *between paragraphs*. Now, what could you do to insert a blank line between the first and the second sentence, that is, within the first paragraph? Well, you could insert yet another `<br>` tag:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Linebreak example 4</title>
    </head>
    <body>
        <p>This is the first sentence.
        <br>
        <br>
        This is the second sentence.</p>

        <p>This is the third sentence.</p>
    </body>
</html>
```

Now it should look like in **Figure 6**. Please be aware that excessive insertion of too many consecutive `<br>` tags sometimes is considered bad style and instead paragraph tags should be used. Please decide for yourself what suits you best.

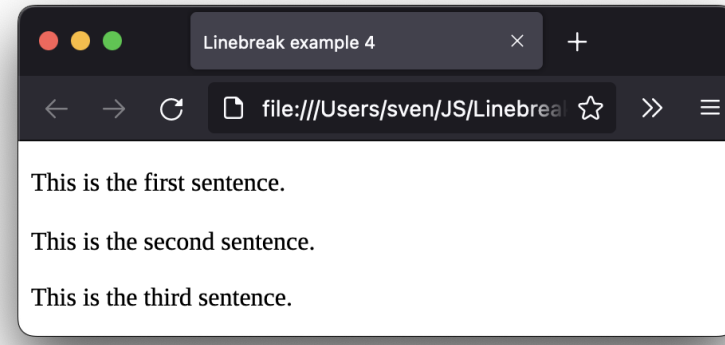


Figure 6. Blank lines between every sentence

Bold, Italics & Co.

The "regular" text without any formatting is displayed in an HTML file exactly as it is displayed in an ordinary text file: all letters have the same font size and style. Fortunately there are various tools to change both font sizes and styles of some parts of the text.

Bold

If you want to display certain words as bold text, just insert the tag `<strong>` before the word where you want the bold print to start. Where you want the bold text to end, insert the closing tag `</strong>`:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Bold text</title>
    </head>
    <body>
```

```
        <p>I want <strong>these three words</strong>
displayed bold.</p>
    </body>
</html>
```

Saved inside the body of a HTML file it will look like **Figure 7**.

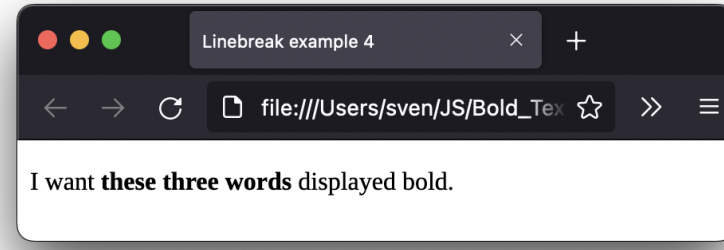


Figure 7. If you look closely you can recognize the bold text

Italics

Just as you did with the bold text, you can use italics for certain parts of the text. Use the tags `<em>` and `</em>` (short for "emphasis") and something similar like **Figure 8** should be the result:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Italics text</title>
    </head>
    <body>
        <p>Now let's make <em>these words</em>
italics.</p>
    </body>
```

```
</html>
```

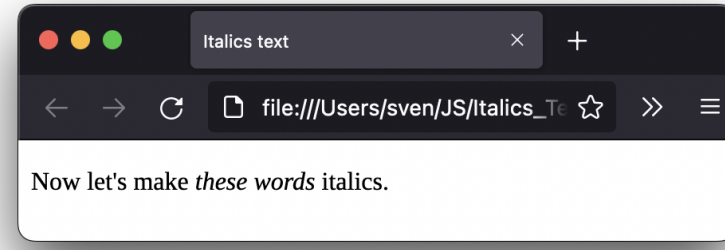


Figure 8. Italics in action

Combined formatting tags

What's better than bold or italics? Well of course, bold *and* italics combined together! You can combine different tags and it will look like in **Figure 9**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Bold italics text</title>
    </head>
    <body>
        <p>Combining <strong><em>bold &
italics</em></strong> text.</p>
    </body>
</html>
```

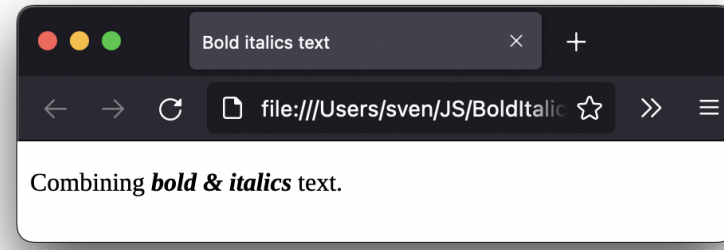


Figure 9. Bolded italics (…or italicized bolds?)

Please make sure that you always write a tag pair *inside another tag pair* when combining tags, i.e. don't mix up the order of the closing tags. The following example would be wrong, because the order of the closing tags does not match the order of the opening tags:

```
<strong><em>some text</strong></em>
```

It should be:

```
<strong><em>some text</em></strong>
```

This way the `<em>` tag pair is fully inside the `<strong>` tag pair.

…and even some more!

There are a quite a few tags that can be used to format text in HTML files. Here is a small overview of the most common tags. Try them out yourself and maybe even combine some of them:

Opening tag	Closing tag	Formatting

`<strong>`	`</strong>`	bold
`<em>`	`</em>`	italics
`<mark>`	`</mark>`	marked text
`<small>`	`</small>`	smaller text
`<sub>`	`</sub>`	subscript
`<sup>`	`</sup>`	superscript

Headings

The main part of an HTML page is mostly regular text, but if *everything* would be regular text only, it could look cluttered and kind of boring. To separate different parts of the text and to emphasize things, headings can be used. In newspapers, thick headings with large characters are used to introduce the text below, and smaller subheadings between paragraphs help to divide the text into different sections and give the reader a general overview. (You can also find quite a few different sized headings in this book).

Heading tags in HTML work similar to the formatting tags we used earlier. To tell the browser where you want your heading to begin, use the opening tag `<h1>`. The closing tag is `</h1>`. If written correctly, you should see something like in **Figure 10**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Heading 1</title>
    </head>
    <body>
        <h1>My BIG heading</h1>
        <p>This is just some regular text.</p>
```

```
    </body>
</html>
```

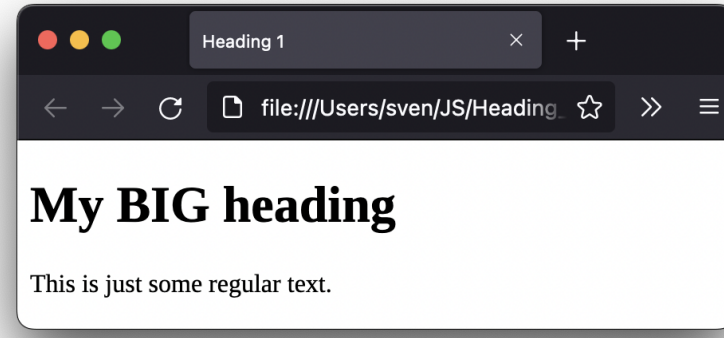


Figure 10. A heading and some regular text below

To write slightly smaller subheadings, just use the smaller variant called `<h2>` instead of `<h1>` like in **Figure 11**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Heading 2</title>
    </head>
    <body>
        <h1>My BIG heading</h1>
        <h2>And a little bit smaller heading 2</h2>
        <p>This is just some regular text.</p>
    </body>
</html>
```

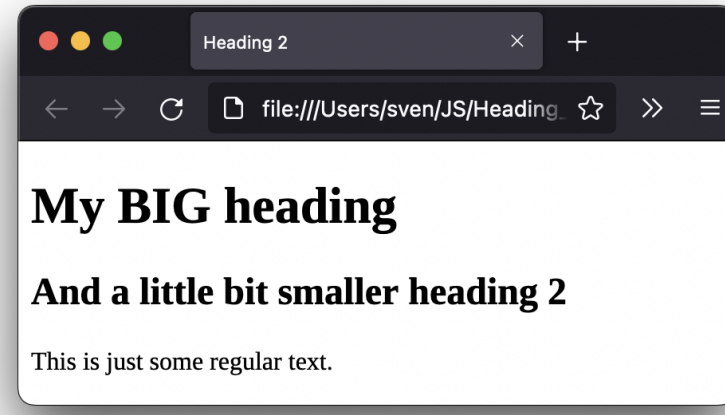


Figure 11. Heading, subheading and regular text

You can make even smaller headings with `<h3>`. *"Wait, does that mean there also are* `<h4>` *and* `<h5>`*?"* Yes. *"And* `<h6>` *too?"* Yes. *"And* `<h7>`*?"* No. It only goes down to `<h6>`. Please don't be disappointed: in a standard HTML file, there is not much visual difference between `<h4>`, `<h5>` and `<h6>`. Here is a short comparison of the different headings. The associated browser screenshot is **Figure 12**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Heading comparison</title>
    </head>
    <body>
        <h1>Heading h1</h1>
        <h2>Heading h2</h2>
        <h3>Heading h3</h3>
        <h4>Heading h4</h4>
        <h5>Heading h5</h5>
```

```
        <h6>Heading h6</h6>
        <p>Regular text</p>
    </body>
</html>
```

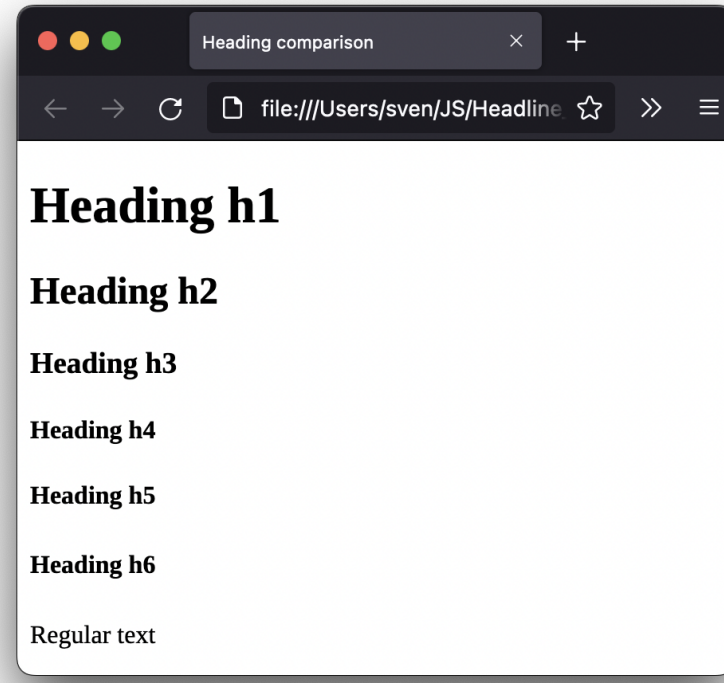


Figure 12. We've got so many headings, they'll blot out the sun! (Hidden movie reference No.5)

Try it out yourself!

To try out what you have learned so far, here is a small task for you: Create a small, fake newspaper report as an HTML page. Use a headline, regular text and sub-headings. You can see a small example at **Figure 13**.

You can choose the topic of this newspaper article freely: write about a real event, or make something up. Just make sure that you don't write something about other people or things in a way that could hurt anybody's feelings. Otherwise, please let your journalistic imagination run wild ;-).

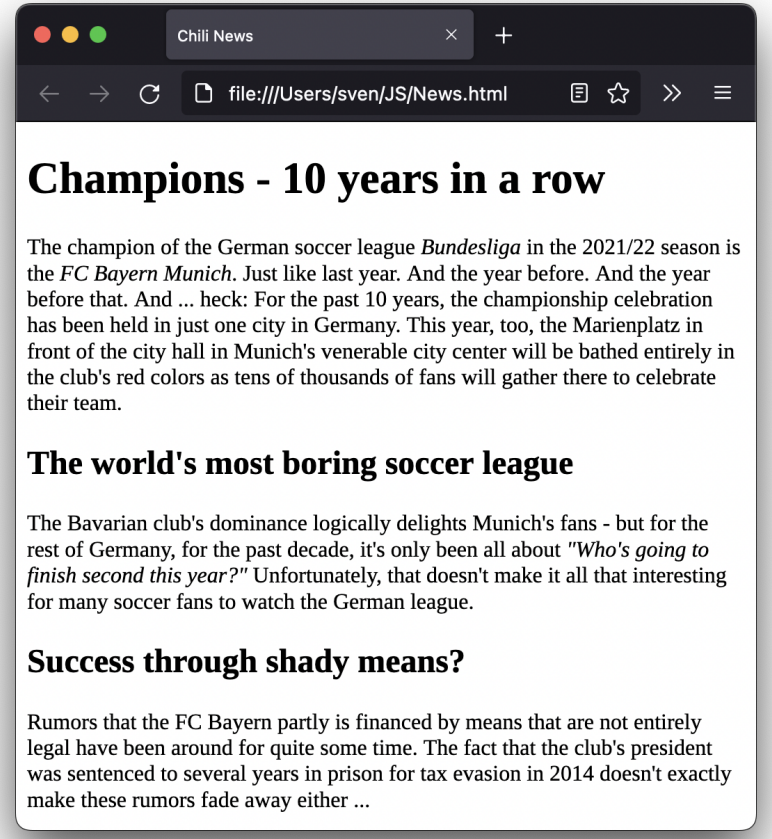


Figure 13. The latest issue of the world famous `Chili News`

Colors

Let's bring some color into our HTML! To color any text, you just need to add the following code:

```
style = "color: Name-of-the-color;"
```

You have to insert this code into the opening tag of the element you want the color to be changed, and write a real color name instead of `Name-of-the-color`. For example, the heading's color in **Figure 14** will be red:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Colors 1</title>
    </head>
    <body>
        <h1 style = "color: Red;">My BIG heading</h1>
        <h2>And a little bit smaller heading 2</h2>
        <p>This is just some regular text.</p>
    </body>
</html>
```

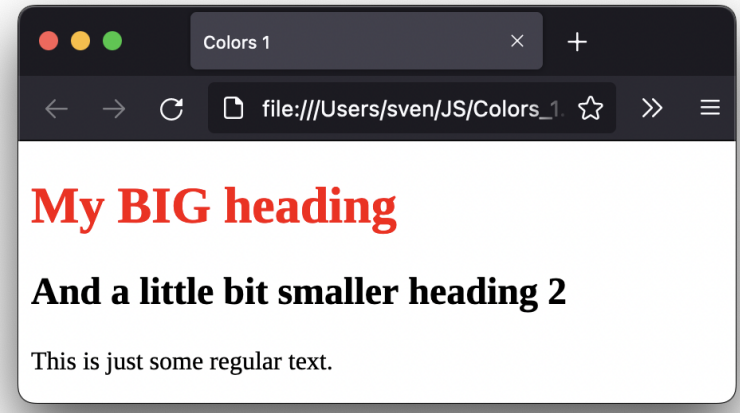


Figure 14. The heading now in red

Let's try it out and use some different colors. The basic color names like `Red`, `Blue`, `Green` or `Yellow` can be used even with ancient browsers. But a bit more modern browsers are able to display a whole bunch of other colors with fancy names like `Blanchedalmond`, `Cornsilk`, `Dodgerblue` or `Firebrick`. Down here are two lists: one with examples of the "old" color names even for ancient browsers, and another one with some additional color names for slightly more modern browsers. There are other ways to display colors in browsers, for example with number codes, but it is much easier to use their names instead. And a HTML code using the color `Tomato` looks so much tastier than specifying this color with the code `#ff6347`, doesn't it?

- **Basic color names:** Black, White, Blue, Green, Maroon, Red, Purple, Teal, Yellow
- **Modern color names:** Blanchedalmond, Cornsilk, Darkblue, Deeppink, Dodgerblue, Goldenrod, Grey, Lightgreen, Midnightblue, Orange, Peru, Saddlebrown, Tomato

For a complete list of HTML color names please see this Wikipedia page: https://en.wikipedia.org/wiki/Web_colors#HTML_color_names

What are you waiting for? Color it up like crazy!

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Colors 2</title>
    </head>
    <body>
        <h1 style = "color: Red;">My BIG heading</h1>
        <h2 style = "color: Blue;">And a smaller heading
2</h2>
        <p style = "color: Orange;">This is just some
regular text.</p>
        <p style = "color: Saddlebrown;">Yet another
paragraph.</p>
    </body>
</html>
```

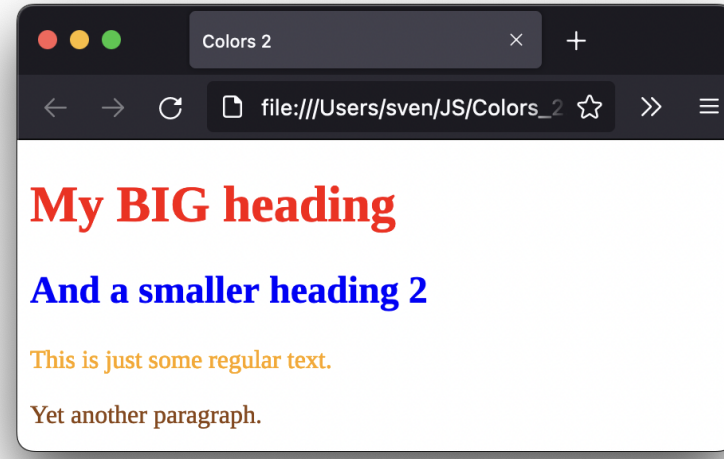


Figure 15. You never can have enough colors...

In the same way that text can be colored, the background of the text can also be colored. The code to use is:

```
style = "background-color: Name-of-the-color;"
```

Here `Name-of-the-color` has to be replaced with a real color name. This code has to be inserted into the opening tag just like we did with the text color. You can even combine these two color specifications - one for the text color and one for the background-color. Just make sure that there is a semicolon `;` between each color specification. In the following example, a semicolon is placed immediately after the color specification `Blue` of the `<h2>` heading, separating it from the background-color specification that is following.

To change the background-color of the whole page, just put this specification inside the opening `<html>` tag or `<body>` tag, like seen in **Figure 16**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Colors 3</title>
    </head>
    <body style = "background-color: Tomato;">
        <h1 style = "color: Red;">My BIG heading</h1>
        <h2 style = "color: Blue; background-color:
Peru;">Another h2</h2>
        <p style = "color: Orange;">This is just some
regular text.</p>
        <p style = "color: Saddlebrown;">Yet another
paragraph.</p>
    </body>
</html>
```

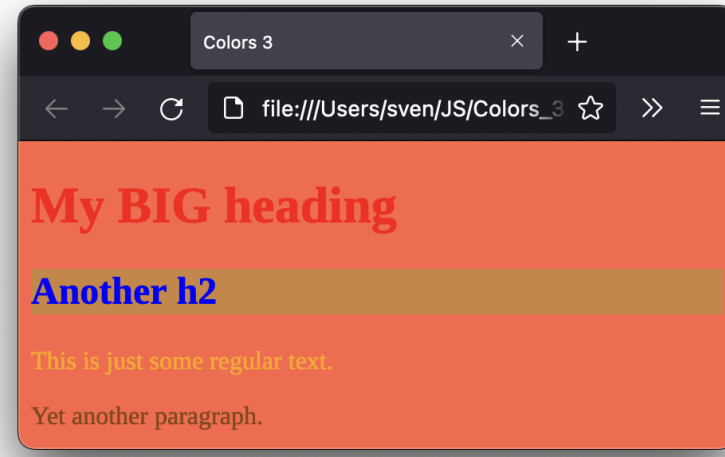


Figure 16. Color explosion in the browser

However, with **Figure 16** you can also clearly see that sometimes

it's not too good an idea to use all the available colors of the palette at once in a HTML page. You have to experiment a bit and see which colors go well together, or which colors combined are not really readable. Sometimes just a little less coloring could get overall nicer results.

Font

You probably already guessed it: just like you can change the colors of your text, you can also change the font of your text. You have a super cool font on your PC and want to use it for your HTML page? No problem, just write in your HTML code that this font should be used. **But!** That *you* have this super cool font installed on your PC does not mean that *all other people* have this super cool font on their device too! Of course, there are ways and means to code your HTML page in such a way that the browser of every user of your HTML page automatically downloads desired fonts from the Internet, but very few users are really happy about fonts being downloaded to their devices every time your HTML page is loaded. (Note also that a not-small amount of users have their browsers set to not download new fonts from the Internet, even if an HTML page explicitly requests it.)

And even if some browsers really download the font you want, it doesn't mean that this font will look as good on their devices as it does on your PC. There is a huge difference between how an HTML page looks on a large computer screen, for example, and how the same page looks on a small smartphone screen … that's why every device has its own special set of pre-installed fonts.

My advice would therefore be not to require a specific font, but only to specify which general font-family should be used. This way you can be sure that your HTML page looks *roughly* the same on every kind of device and still looks good.

If you are absolutely desperate to use a special, super cool font for the title of your HTML page, for example, it might be better and easier to display an image of the text that uses this font instead of letting users download the entire font. How to display images in HTML pages will be explained a few pages later in the book.

Generally speaking, there are three general font-families:

- **Serif**: The font-family used in this book. This font-family has additional small strokes on most letters. Print media such as newspapers like to use it.
- **Sans-serif**: This font family has no additional small strokes on its letters. Most computers and smartphones use it to display their menus and dialogs.
- **Monospace**: Here, each letter takes up just as much horizontal space as every other letter. The HTML code in this book is displayed with a monospace font-family.

You can specify which font-family to use, similar to how colors are specified, by inserting the following code into an opening tag:

```
style = "font-family: Name-of-the-font-family;"
```

Replace `Name-of-the-font-family` with `serif`, `sans-serif` or `monospace` to get a result like **Figure 17**. In general, it is better not to use too many different fonts or font-families mixed wildly throughout a HTML page - it will look strange and cluttered.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

```
        <title>Font-families</title>
    </head>
    <body>
        <h1 style = "font-family: serif;">My serif
heading</h1>
        <h1 style = "font-family: sans-serif;">My sans
heading</h1>
        <h1 style = "font-family: monospace;">My
monospace heading</h1>
    </body>
</html>
```

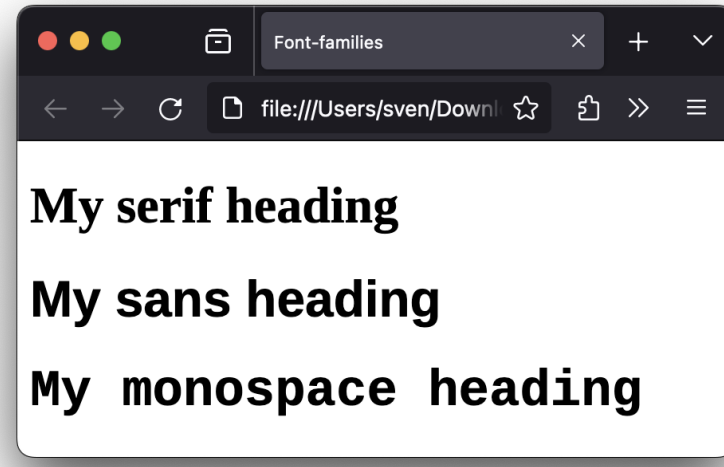


Figure 17. The honorable three font-families

If you don't plan to change the fonts of your HTML page often, it would be an idea to specify the font inside the opening `<html>` tag or `<body>` tag (the same way we did with the background color). You can also do this in combination, like for example:

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <meta charset="utf-8">
        <title>(code example)</title>
    </head>
    <body style = "background-color: Peru; font-family:
sans-serif;">

    </body>
</html>
```

When specifying different styles, please make sure that there is a semicolon `;` between each style specification. In the previous example, a semicolon is placed immediately after the background-color specification `Peru`, separating it from the font-family specification that is following.

Links

The great thing about HTML pages is that you can click on links and be redirected immediately. The even greater thing is that links can be created quite easily. First you write the opening link tag `<a>`, and right after that you write the corresponding closing tag `</a>` :

```
<a> </a>
```

Within the opening tag you add the code `href = " "`. Inside the quotes of this code you then write the net address you want to link to.

```
<a href = "https://en.wikipedia.org"> </a>
```

Then, *between the two link tags* you write the text which should be clicked as a link on the HTML page. This can be the same text as the link address, but it can also be something completely different:

```
<a href = "https://en.wikipedia.org">Link to
Wikipedia</a>
```

In many browsers the link is then underlined and displayed in a different color than the regular text, as can be seen for example in **Figure 18**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Link example</title>
    </head>
```

```
    <body>
        <p>This is just regular text. Nothing
special.</p>
        <p>But here we've got a <a href =
"https://en.wikipedia.org">Link to Wikipedia</a>, how
cool is that!</p>
    </body>
</html>
```

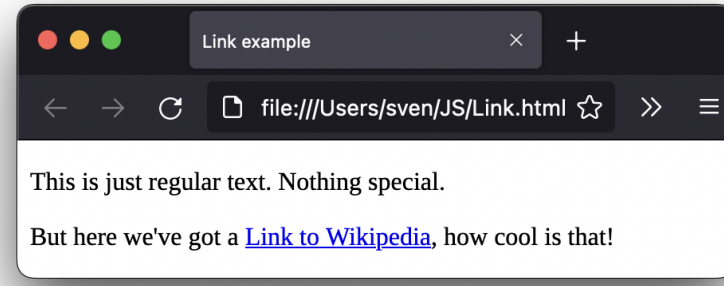


Figure 18. A link to unlimited knowledge

Links to locations on the same page

You can link not only to other pages on the Internet, but also to places on the same page. For example, if you have a fairly large website with a lot of text - similar to a newspaper website - you can create a kind of "table of contents" at the top of your website. If you then click on these individual links, you will jump to the corresponding place further down on the website.

Creating these links on the same page actually works in exactly the same way as creating links to other websites - you just have to specify an additional target point on the website to which you want to jump, so to speak, when you click on a link. This target

point is called an "anchor".

Let's create a small example page with two links and two anchors! We'll start with a very simple page containing two headlines and two paragraphs of text:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Anchor</title>
    </head>
    <body>
        <h1>Headline No.1</h1>
        <p>...some text...</p>

        <h1>Headline No.2</h1>
        <p>...some more text...</p>
    </body>
</html>
```

Now we want to insert two links at the top of this website: one link per headline. If you click on the first link, you will jump to the first headline. If you click on the second link, you jump to the second headline.

To do this, we first need to add an anchor to each headline so that the browser knows where to jump to when the link is clicked.

An anchor is quite simple to create. You simply add a so called "id" inside the opening tag `<h1>` of the headline:

```
        <h1 id = "MyAnchor1">Headline No.1</h1>
```

You can give this anchor id any name you'd like. In the example it

is simply named "MyAnchor1". The second headline gets its own anchor id as well, so the website code will look like this:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Anchor</title>
    </head>
    <body>
        <h1 id = "MyAnchor1">Headline No.1</h1>
        <p>...some text...</p>

        <h1 id = "MyAnchor2">Headline No.2</h1>
        <p>...some more text...</p>
    </body>
</html>
```

You do not necessarily have to select a headline as an anchor, but can actually insert your anchor anywhere you want on the website. However, the website will be easier to use for the users if concise parts of the website are selected as anchors, such as headlines, images or similar.

Now only the actual links need to be inserted. In our example we insert them at the top of the website. These links are created in the same way as links to other webpages, with the only difference that the respective anchor on the website is now specified as the target. Please note that *in the link* an anchor as target has to be written with a # character in front:

```
<a href = "#MyAnchor1">Click here to go to Headline
No.1</a>
```

The text `Click here to go to Headline No.1` is displayed on the webpage and when being clicked, the browser will jump to `MyAnchor1` (the `#` sign meaning the anchor is somewehere on that webpage).

The whole HTML code could look like the following, while in the browser it will look like **Figure 19**:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Anchor</title>
        <meta charset="utf-8">
    </head>
    <body>
        <a href = "#MyAnchor1">Click here to go to
Headline No.1</a>
        <br>
        <a href = "#MyAnchor2">Click here to go to
Headline No.2</a>

        <h1 id = "MyAnchor1">Headline No.1</h1>
        <p>...some text...</p>

        <h1 id = "MyAnchor2">Headline No.2</h1>
        <p>...some more text...</p>
    </body>
</html>
```

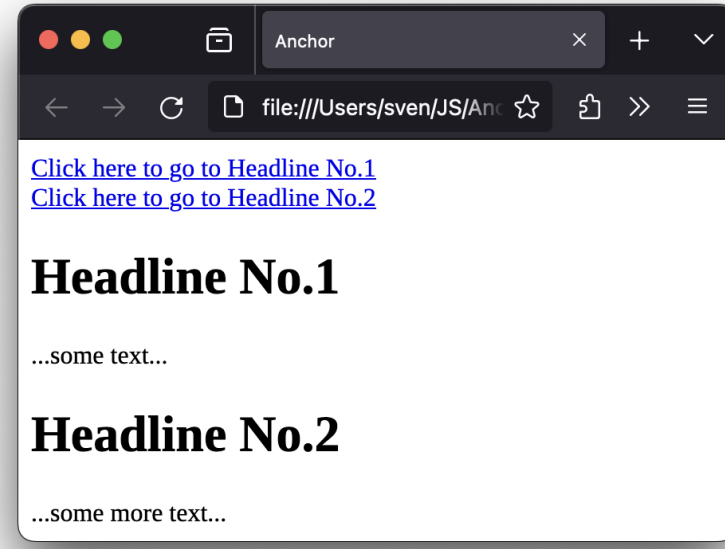


Figure 19. Links (and anchors) on the same webpage

If you are using a fairly large computer monitor and click on the links described above, there will probably not be much movement: if the screen is so large that the entire HTML page is already displayed from the outset, then jumping to the HTML links on the same page will not be noticeable. To really "jump" to the clicked links, you would either have to use a smaller screen, or simply write more text at the paragraphs where we wrote `"···some text···"` and `"···some more text···"`.

Links to other files

Linking to files basically works in the same way as linking to other websites.

Let's say you have a file called "MyData.pdf" that you want to link to on your HTML page. If this PDF file is in exactly the same folder as your HTML file, the link will look like this: `<a href="MyData.pdf"> Text-to-be-clicked</a>` . Inside the quotation marks after `href=` the name of the file has to be written. The text between the `<a>` and `</a>` tags will be displayed in a special color and also be underlined in the browser. If this underlined text is clicked in the browser, the download of the file will start.

It could look like **Figure 20**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Link example 2</title>
    </head>
    <body>
        <p>Regular text. <a href="MyData.pdf">Click
here!</a> More regular text.</p>
    </body>
</html>
```

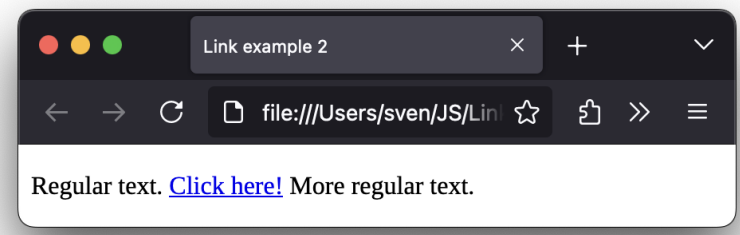


Figure 20. Linking to files is just as easy

If the file is **not** located in the same folder as your HTML file, but

for example in a subfolder called "MyFiles", you must adapt the code accordingly:

```
...
<p>Regular text. <a href="MyFiles/MyData.pdf">Click
here!</a> More regular text.</p>
...
```

This means that the folder "MyFiles" is in the same folder as your HTML file, and inside that folder "MyFiles" the data to download is located.

Of course you can also link to other files that are not located on your computer but somewhere on the web. In such a case you have to write the whole URL (i.e. the "internet adress" of that file) into your code. For example, the download link to this textbook would look like this:

```
...
<p>Regular text. <a
href="http://sven.kir.jp/JS/Learn_HTML_the_fun_way.pdf">C
lick here to download the book.</a> More regular
text.</p>
...
```

You can find a more detailed description of what you should bear in mind when linking files and what the so called "folder hierarchy" is all about in the chapter *Sharing & uploading your HTML files*.

Images

Various images and graphics can be inserted into HTML pages. The code for this is just a little bit more extensive than the code earlier for the insertion of links, but also not really *that* much more complicated.

To keep it simple and to make sure that on most devices and browsers these images really can be displayed, it is a recommendation to use only images in `.jpg` (sometimes written as `.jpeg`) or `.png` formats. There are other image formats that *may* still be displayed as well, but with the two image formats described above you can be pretty sure that graphical browsers will be able to display them without problems and without long loading times almost no matter what device the user is using.

To insert an image into your HTML page, all you need to do is inserting the `<img>` tag into your HTML code. The `<img>` tag is stand-alone, i.e. there is *no* closing `</img>` tag:

```
<img>
```

Inside the `<img>` tag, in the simplest version, you then only need to specify where this image is stored, using `src` (which stands for "source"):

```
<img src = " ">
```

Inside the quotation marks you write the location of the image. This could for example be an Internet address, but in many cases the image is stored where the HTML file is stored. If the image is stored together with your HTML file *in the exact same folder*, you only need to specify the image name inside the quotation marks.

For example, the code to display an image named `logo.png` is:

```
<img src = "logo.png">
```

A small example HTML page with a heading, a small image for the logo and some text would be like the following as shown in **Figure 21**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Image example 1</title>
    </head>
    <body>
        <h1>CHILI RULEZ!</h1>
        <p><img src = "logo.png"></p>
        <p>Welcome to my chili page.</p>
    </body>
</html>
```

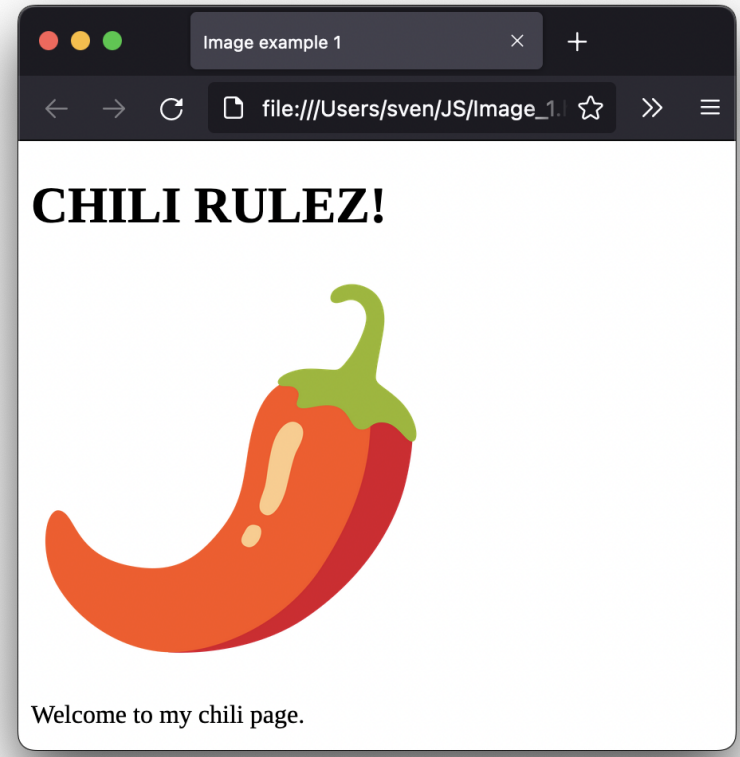


Figure 21. Headline, some text and a monster chili

Whoa, that image is *way* too big, it's even bigger than the heading! It should have been a small logo and not a whole page chili attack. Well, that's because the browser usually displays images in their original size. If you have a rather large image, this can become a problem. One solution could be to reduce the size of the image using an image editing software. Another, simpler solution would be to specify in the HTML code how large the image should be displayed. The code for this is

```
width = "Width-as-a-number"
```

if you want to specify the image width, or

```
height = "Height-as-a-number"
```

if you want to specify its height. You will have to replace `Width-as-a-number` or `Height-as-a-number` with the preferred image size and then simply write this code into the `<img>` tag. The result will be something like in **Figure 22**. The size of images in HTML pages is usually specified using `px` (which stands for "pixel"), but there also are other ways of specifying the size. For the sake of simplicity, `px` is used throughout this book. Sometimes you have to experiment a little bit and see what image sizes work best for your HTML page.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Image example 2</title>
    </head>
    <body>
        <h1>CHILI RULEZ!</h1>
        <p><img src = "logo.png" width = "25px"></p>
        <p>Welcome to my chili page.</p>
    </body>
</html>
```

Please pay attention here! With the formatting of text we did before, several specifications were always separated by a semicolon `;` . But now within the `<img>` tag, different specifications are simply separated by a space.

Figure 22. Now that size is more kind of a logo

And always remember: that's how the HTML page is displayed on *your* screen - different devices with smaller or bigger screens could display the HTML page and its image with different dimensions.

You could store all images in exactly the same folder on your PC as the HTML file, but if you have a lot of images it can quickly become crowded. A good idea would be to create a subfolder in the folder where the HTML file is stored, e.g. named `images`, and store all images there. So if you saved your HMTL files in the folder `JS`, then put all related images into the folder `images`.

For example, if you have an image named `logo.png` in the subfolder `images`, the specification in the HTML code would be the following:

```
<img src = "images/logo.png">
```

The location of images is always specified *relative* to the location of the HTML page. A little example to explain this: In your folder `JS`, where your HTML file is located, you created a subfolder named

`stuff` where you want to put all the things that will be used by your HTML file. And inside this `stuff` folder you have created yet another subfolder named `pics` where you want to save all your images. The HTML code to display an image from this "sub-subfolder" would look like the following:

```
<img src = "stuff/pics/logo.png">
```

Somehow the chili logo on the example HTML page still doesn't look really good ... the logo is quite far away from the heading. Now what could be done to display the logo in the same line as the heading? Think for yourself if you can find an answer, you have 5 seconds. *5, 4, 3, 2, 1, time up!* Okay, one possibility would be to write the `<img>` tag between the two `<h1>` tags. And why only one logo? Let's add two logos right away - of course with reduced sizes like in **Figure 23**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Image example 3</title>
    </head>
    <body>
        <h1> <img src = "logo.png" width = "25px"> CHILI
RULEZ! <img src = "logo.png" width = "25px"></h1>
        <p>Welcome to my chili page.</p>
    </body>
</html>
```

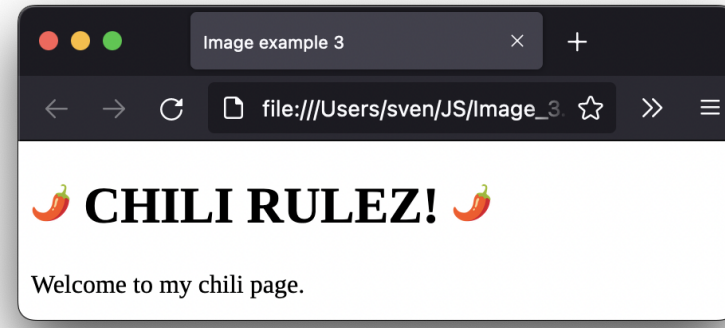


Figure 23. That heading looks `hot` (...Sorry, that pun was really cheap)

Image border and alt name

You can do some more with images. How about a nice border around the image for example? That's quite easy, the code for it is

```
border = "Thickness-as-a-number"
```

where instead of `Thickness-as-a-number` you specify the desired thickness of the border with `px`. And while we are at it: with the code

```
alt = "Text-you-want-for-alt"
```

a description for the image can be provided replacing `Text-you-want-for-alt` with some text. This description is normally not displayed in the browser. *"Hmm, if this description is not displayed, why do you need it at all?"* If there are problems loading that image, this alternative description will instead be displayed. Browsers that do not display graphics will also display this

description. And browsers for people with visual impairment can use it to describe it to its users. It has quite its advantages to use this kind of image description.

Let's extend the example from before a bit further and add an image with border and some alternative descriptions for all the images used like in **Figure 24**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Image example 4</title>
    </head>
    <body>
        <h1> <img src = "logo.png" width = "25px" alt =
"Chili logo"> CHILI RULEZ! <img src = "logo.png" width =
"25px" alt = "Chili logo"></h1>
        <p>Welcome to my chili page.</p>
        <p><img src = "images/bowl.png" width ="100px"
alt="Chili bowl" border="5px"></p>
        <p>A selection of the best chili receipts on the
planet.</p>
    </body>
</html>
```

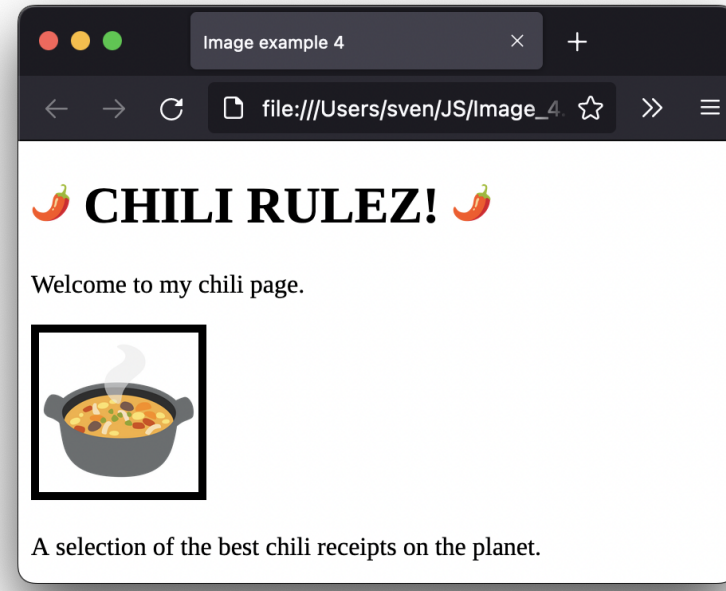


Figure 24. Border around an image

Try it out yourself!

Now it's time to show your skills. Make an HTML page about a hobby of your choice: sports, food, vehicles ... whatever you like. On that page, describe what your chosen hobby is about and add one or even more pictures to explain. Also make a heading, maybe even with a logo? And paint everything in matching *(but still readable!)* colors.

Styling the body

All HTML examples until now had their content on the left side. But this can be changed. One possibility would be to center the entire page content. To do this, you could for example simply insert the code

```
style = "text-align: center;"
```

into the opening `<body>` tag, as seen in **Figure 25**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Center test</title>
    </head>
    <body style = "text-align: center;">
        <h1>Heading</h1>
        <p>Regular text.</p>
        <p><img src = "images/bowl.png" width ="100px"
alt="Wrap"></p>
    </body>
</html>
```

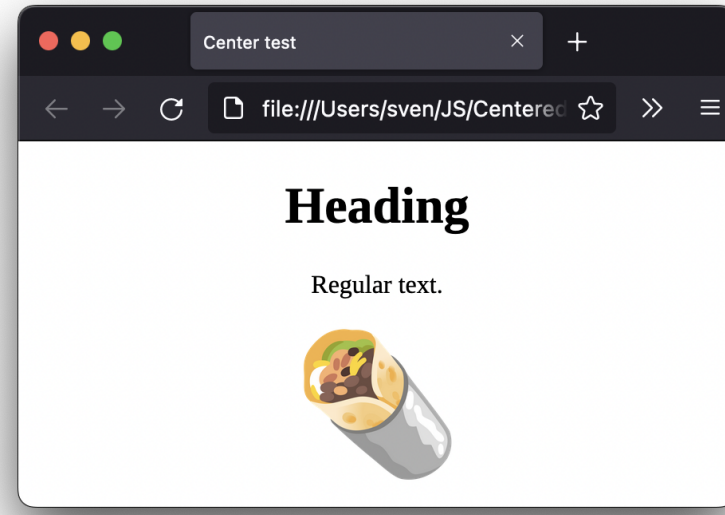


Figure 25. Everything centered

However, if you have a lot of regular text on the page, it could get a bit awkward to read when everything is centered in the middle. You would have to somehow find a way to still have the page content left-aligned, but not quite so close to the left page margin. One way to do this would be to "shrink" the `<body>` just a bit.

To illustrate this in an easy to understand way, first we specify two different background colors for the opening `<html>` tag and the opening `<body>` tag respectively:

```
<!DOCTYPE html>
<html style="background-color: Tomato;">
    <head>
        <meta charset="utf-8">
        <title>Center test 2</title>
    </head>
    <body style="background-color: White;">
        <h1>Heading</h1>
```

```
        <p>Regular text.</p>
        <p><img src = "images/wrap.png" width ="100px"
alt="Wrap"></p>
    </body>
</html>
```

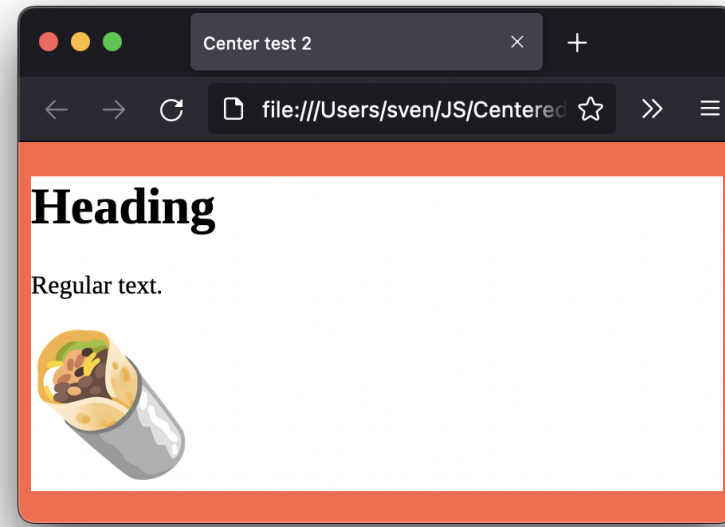


Figure 26. <html> and <body> with different background-colors

In **Figure 26** you can see exactly where the border between <html> and <body> is on the HTML page. Now you only have to make the <body> part a bit smaller horizontally. This can be done by specifying the width in the opening <body> tag. For this, width can be used to specify the width of the element in question. You could specify the size of this width with px, e.g.

```
<body style="width: 550px;">
```

and the <body> would then always be exactly 550 pixels wide, no

matter how big the screen or browser window of the used device is. If you specify the size with `%` instead of `px`, for example

```
<body style="width: 75%;">
```

then the width of the `<body>` will be just that much percentage of the browser window. For example, if you specify

```
<body style="width: 50%;">
```

the `<body>` will be exactly half the size of the browser window's width. Feel free to experiment a bit with a higher or lower percentage and see how the HTML page will look like, maybe something similar to **Figure 27**:

```
<!DOCTYPE html>
<html style="background-color: Tomato;">
    <head>
        <meta charset="utf-8">
        <title>Center test 3</title>
    </head>
    <body style="background-color: White; width: 75%;">
        <h1>Heading</h1>
        <p>Regular text.</p>
        <p><img src = "images/wrap.png" width ="100px"
alt="Wrap"></p>
    </body>
</html>
```

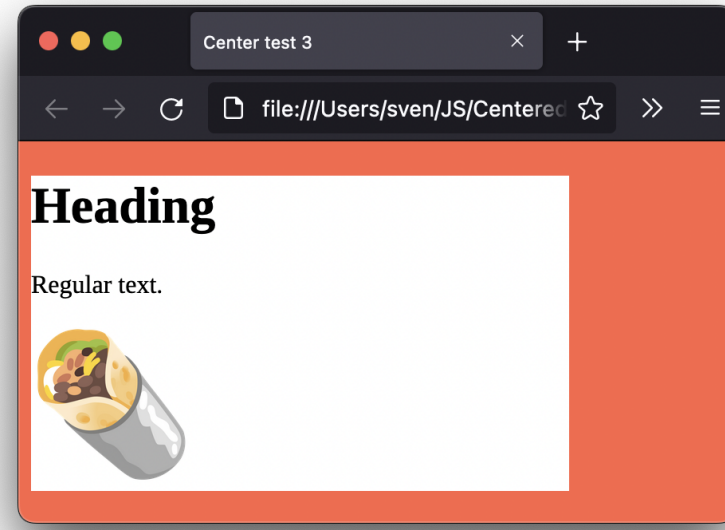


Figure 27. The <body> now shrinked horizontally

Hmm, at **Figure 27** you can now see that the <body> has become smaller horizontally, but it is still all the way to the left. One way to bring the <body> to the center of the HTML page is the following trick: add the code

```
margin: 25px auto
```

to the opening <body> tag. The margin code specifies the space outside around the respective element and actually needs *two* size specifications, e.g.

```
<body style="margin: 25px 50px;">
```

The first size specifier indicates how much space to add at the top and bottom, the second size specifier indicates the space on the left

and right. So with the code example above you would have 25 pixels of space above and below the `<body>`, and 50 pixels each on the left and right. And here comes the trick: if you write `auto` instead of the second size specification, *all* horizontal space on the sides of the `<body>` will automatically be distributed evenly left and right, no matter what size the browser window has, like in **Figure 28**:

```
<!DOCTYPE html>
<html style="background-color: Tomato;">
    <head>
        <meta charset="utf-8">
        <title>Center test 4</title>
    </head>
    <body style="background-color: White; width: 75%;
margin: 25px auto;">
        <h1>Heading</h1>
        <p>Regular text.</p>
        <p><img src = "images/wrap.png" width ="100px"
alt="Wrap"></p>
    </body>
</html>
```

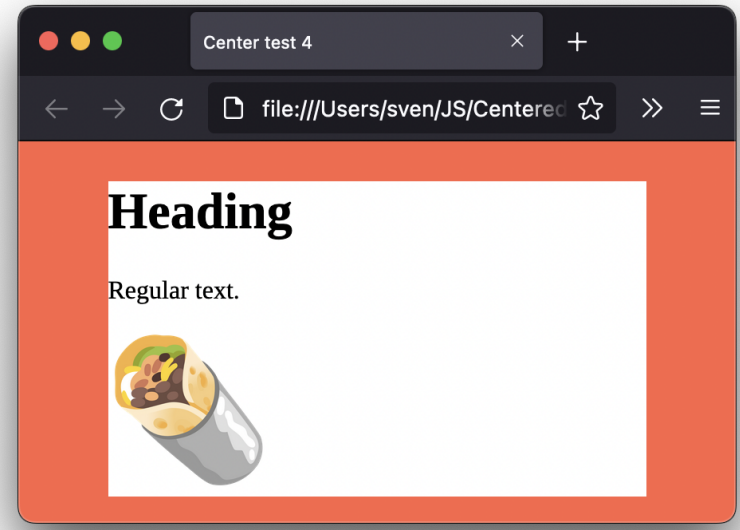


Figure 28. Now we got the `<body>` centered

Okay, now the body itself is in the center, but there is one not so nice thing left: the text is a little bit too close to the left body margin. This really stands out when the backgroud-colors of `<html>` and `<body>` are different. But don't worry, there is a code for that too. With

```
padding: 15px
```

you can add some space between the content and the inner margin. Or in other words: while the code `margin` we were using in the example above specifies the distance *outside around an element*, the code `padding` specifies the distance *inside an element to the element border*.

Please experiment yourself, if more or less than `15px` will give you a better result like in **Figure 29**:

```
<!DOCTYPE html>
<html style="background-color: Tomato;">
    <head>
        <meta charset="utf-8">
        <title>Center test 5</title>
    </head>
    <body style="background-color: White; width: 75%;
margin: 25px auto; padding: 15px;">
        <h1>Heading</h1>
        <p>Regular text.</p>
        <p><img src = "images/wrap.png" width ="100px"
alt="Wrap"></p>
    </body>
</html>
```

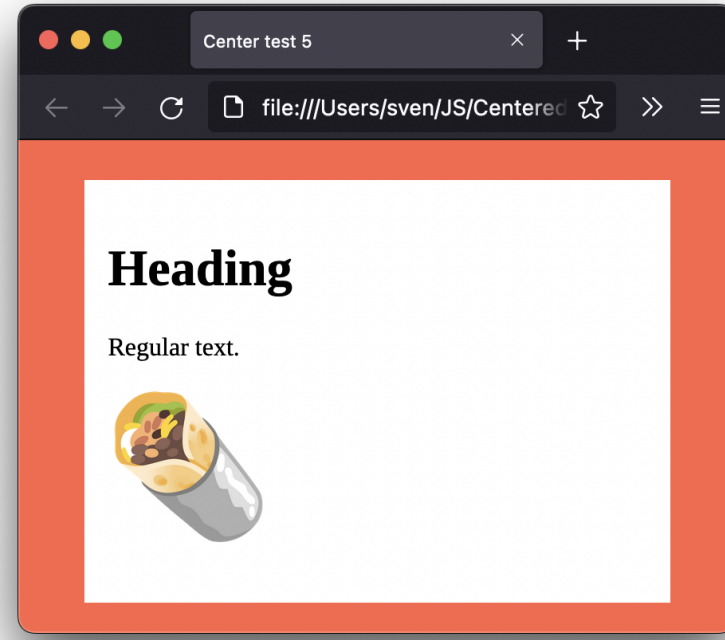


Figure 29. Padding inside the <body>

Now, if you really wanted to nitpick, you could say that the heading would look better if it were in the middle and not at the left margin. And the image should be in the middle too. Doing that is actually pretty easy, can you figure out how?

One way to center the heading would be to insert the code

```
style = "text-align: center;"
```

into the opening `<h1>` tag. And to center the image, you could do exactly the same: you only have to insert that code into the opening tag of the `<p>` Paragraph the image is located in, as seen in **Figure 30**:

```
<!DOCTYPE html>
<html style="background-color: Tomato;">
    <head>
        <meta charset="utf-8">
        <title>Center test 6</title>
    </head>
    <body style="background-color: White; width: 75%;
margin: 25px auto; padding: 15px">
        <h1 style = "text-align: center;">Heading</h1>
        <p>Regular text.</p>
        <p style = "text-align: center;"><img src =
"images/wrap.png" width ="100px" alt="Wrap"></p>
    </body>
</html>
```

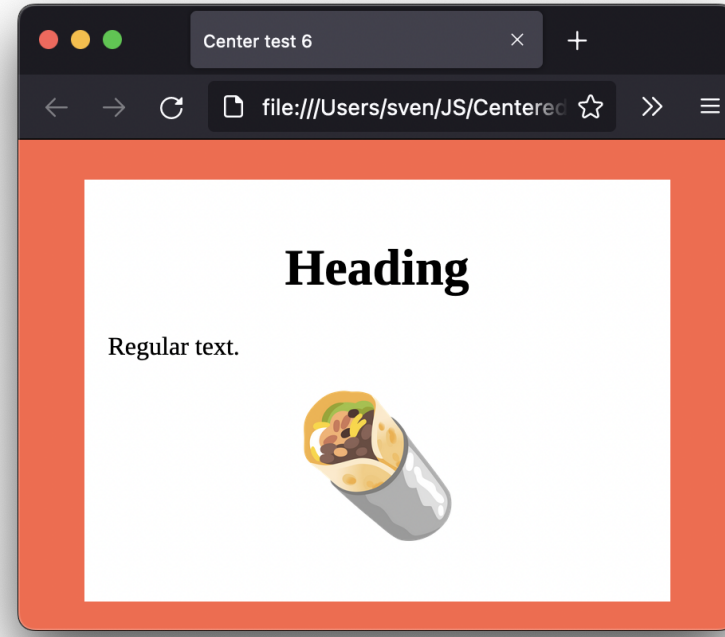


Figure 30. Heading and image centered, while the regular text is left-aligned

Body border & shadow

In an example earlier in this book, a border was created around an image. Not only around images, but around the body you can create a border, too. The codes for this are similar, but please be careful: there are small differences here! To create a 5 pixel wide border around an *image* like earlier in this book, the code is

```
border ="5px"
```

To create a 5 pixel wide border around the *body*, the code is

```
border: 5px solid Black;
```

This code is inserted added to the `style` part inside the opening body tag. `5px` specifies the width of the border in pixels. `Solid` simply means that the border should not be transparent. And you probably already guessed it: `Black` specifies the color of the border. Here at **Figure 31** is an example:

```
<!DOCTYPE html>
<html style="background-color: Tomato;">
    <head>
        <meta charset="utf-8">
        <title>Body border 1</title>
    </head>
    <body style="background-color: White; width: 75%;
margin: 25px auto; padding: 15px; border: 5px solid
Black;">
        <h1 style = "text-align: center;">Heading</h1>
        <p>Regular text.</p>
    </body>
</html>
```

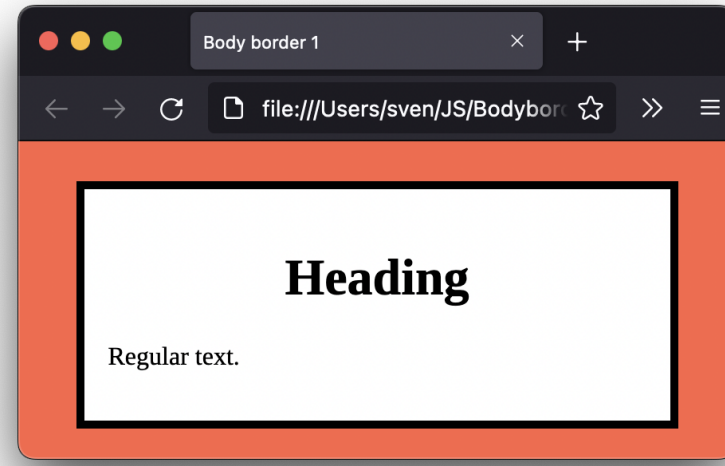


Figure 31. Now the body has some nice border around it

Without further specifications, the border around the body has normal, angular corners. But with a small code addition these corners can be rounded. The code for this would be

```
border-radius: 25px;
```

Please experiment yourself how the radius of the rounded corners changes if you specify a higher or lower number than the 25 pixels in this example code. You can see the rounded border corners with the size of 25 pixels in **Figure 32**:

```
<!DOCTYPE html>
<html style="background-color: Tomato;">
    <head>
        <meta charset="utf-8">
        <title>Body border 2</title>
    </head>
    <body style="background-color: White; width: 75%;
```

```
margin: 25px auto; padding: 15px; border: 5px solid
Black; border-radius: 25px;">
        <h1 style = "text-align: center;">Heading</h1>
        <p>Regular text.</p>
    </body>
</html>
```

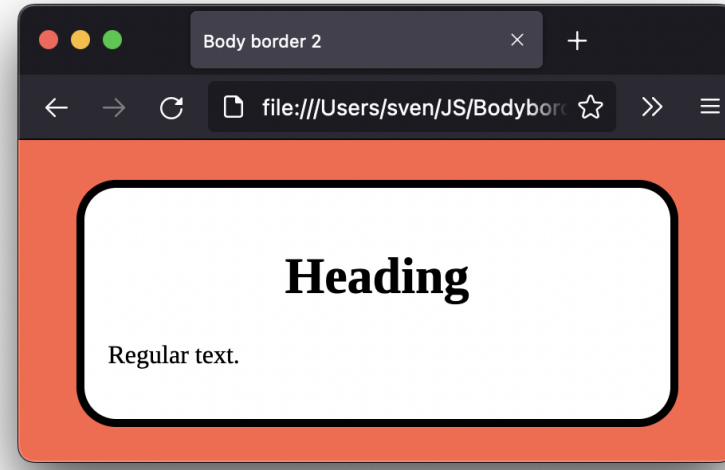


Figure 32. Rounded corners give a softer look

And to make the body stand out a bit, a shadow can be added. The code for this is

```
box-shadow: 10px 10px;
```

where the first pixel specification indicates the offset of the shadow to the right, and the second indicates the offset down. So for example the code

```
box-shadow: 30px 5px;
```

would create a shadow that goes 30 pixels to the right and 5 pixels down. You can see an example with a 10 pixel shadow in **Figure 33**:

```
<!DOCTYPE html>
<html style="background-color: Tomato;">
    <head>
        <meta charset="utf-8">
        <title>Body border 3</title>
    </head>
    <body style="background-color: White; width: 75%;
margin: 25px auto; padding: 15px; border: 5px solid
Black; border-radius: 25px; box-shadow: 10px 10px;">
        <h1 style = "text-align: center;">Heading</h1>
        <p>Regular text.</p>
    </body>
</html>
```

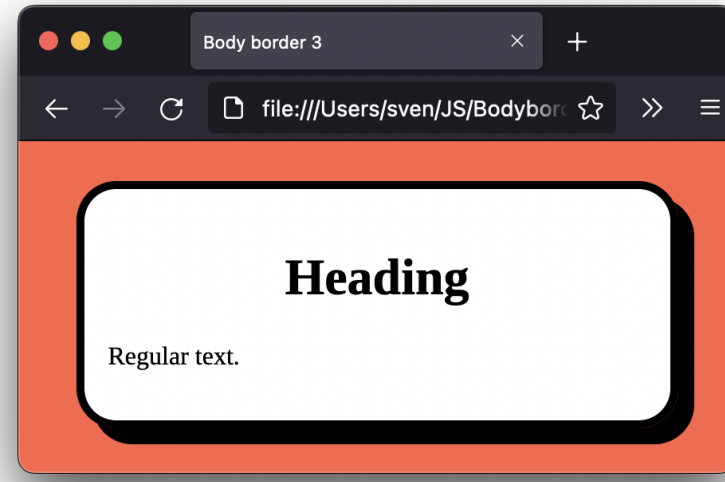


Figure 33. With a shadow added, the body seems to stands out

Horizontal line

To create a separator line on your HTML page, simply add the standalone tag `<hr>` . This line can be anywhere, for example between a heading and some text, before or after an image, or between two paragraphs of regular text as in **Figure 34**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Horizontal line 1</title>
    </head>
    <body>
        <h1>Heading</h1>
        <p>Some regular text.</p>
        <hr>
        <p>More regular text.</p>
```

```
    </body>
</html>
```

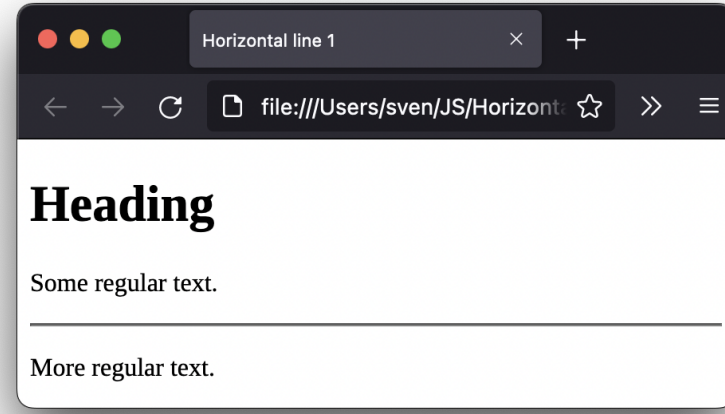


Figure 34. A horizontal line to separate stuff

Without further specification the horizontal line goes over the whole width of the browser window. With the additional specification

```
<hr width = 50%>
```

the width can be specified. Here in this example, the horizontal line would be half the size of the browser window. If no other changes to the style are specified, the horizontal line will be centered, like in **Figure 35**. Please experiment by yourself how other line widths will look:

```
<!DOCTYPE html>
<html>
    <head>
```

```
        <meta charset="utf-8">
        <title>Horizontal line 1</title>
    </head>
    <body>
        <h1>Heading</h1>
        <p>Some regular text.</p>
        <hr width = 50%>
        <p>More regular text.</p>
    </body>
</html>
```

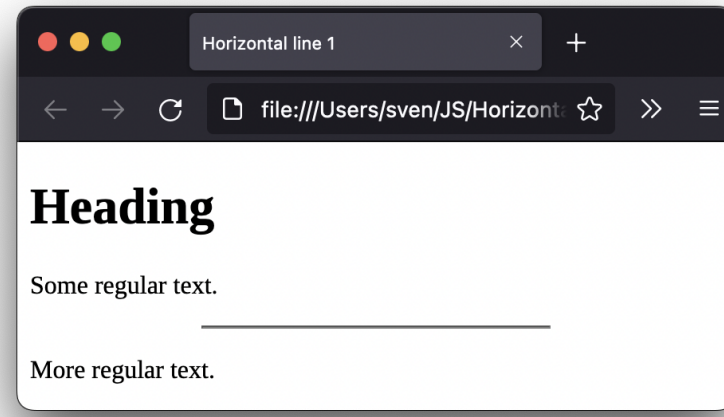


Figure 35. The horizontal line size reduced

Lists

To create a list with multiple items, it doesn't take too much new coding. Creating lists basically works the same way we've been creating normal paragraphs all along. With lists you just don't start each list item with the opening tag `<p>`, but instead with `<li>` (which stands for "list item"). And what is the closing tag for a list item? Exactly, it is `</li>` . Only one additional part has to be added:

all the items of the list have to be inside the list tags `<ul>` and `</ul>` (which stands for "unordered list").

As an example, let's make a list of various ingredients needed for a mild and noble gourmet dish, as in **Figure 36**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>List 1</title>
    </head>
    <body>
        <p>Gourmet meal ingredients:</p>
        <ul>
            <li>Salt, pepper, some olive oil & water</li>
            <li>Tomatoes</li>
            <li>Large onion</li>
            <li>Three or more (...better more!) cloves of
garlic</li>
            <li>A bunch of Jalapeno chilies</li>
            <li>Beans, lots of beans</li>
        </ul>
    </body>
</html>
```

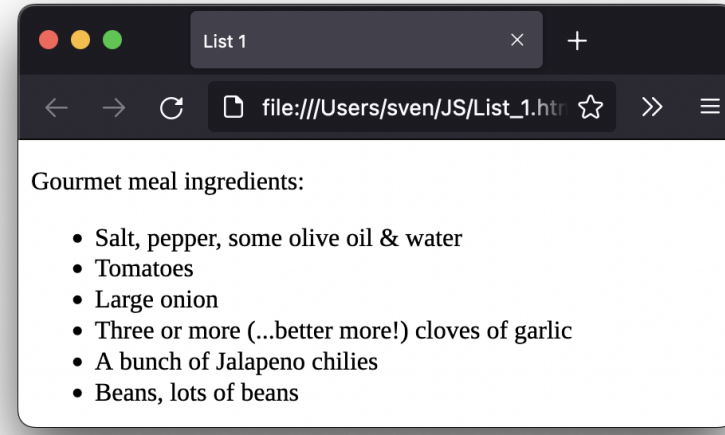


Figure 36. A list for true gourmets

In the above code, if you enclose the list with the tags `<ol>` (which stands for "ordered list") instead of the tags `<ul>` , you will get a numbered list as in **Figure 37**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>List 2</title>
    </head>
    <body>
        <p>Gourmet meal ingredients:</p>
        <ol>
            <li>Salt, pepper, some olive oil & water</li>
            <li>Tomatoes</li>
            <li>Large onion</li>
            <li>Three or more (...better more!) cloves of
garlic</li>
            <li>A bunch of Jalapeno chilies</li>
            <li>Beans, lots of beans</li>
```

```
        </ol>
    </body>
</html>
```

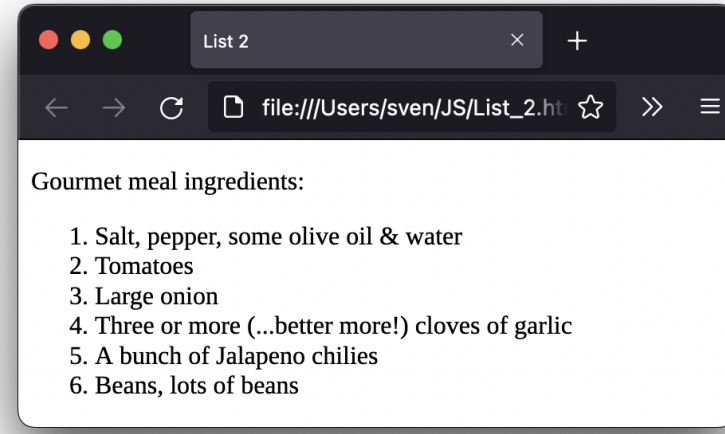


Figure 37. The same list, now numbered

That's enough HTML for today

There is of course much more to HTML than has been covered so far in this book. For absolute programming beginners, however, this should be enough to be able to create attractive HTML pages, and later, with a little JavaScript, some quite nice apps. If you want to learn more about HTML, you should take a closer look at the further mentioned teaching materials in the chapter *Further reading*.

And lastly a hint that might be important for programming beginners: even veteran programmers use Internet search engines to look up how to write code or how to solve coding problems. In fact, they do that very often. *Very, very often!* But this also means

that you don’t necessarily have to learn every last detail of a programming language by heart. A rough overview of the programming language is enough if you also know how to search in textbooks or the Internet to solve your coding problems.

Try it out yourself!

As the final challenge of this book, the task is to create a *"Wanted!"* poster like you might know from old cheesy movies. You can see an example of this poster at **Figure 38**. Use regular text, headings, lists and more of what has been learned so far. (Everything used in the **Figure 38** example has already been presented somewhere in this book!) And as always, it’s best not to make this poster about real people, as they may be unintentionally hurt by this.

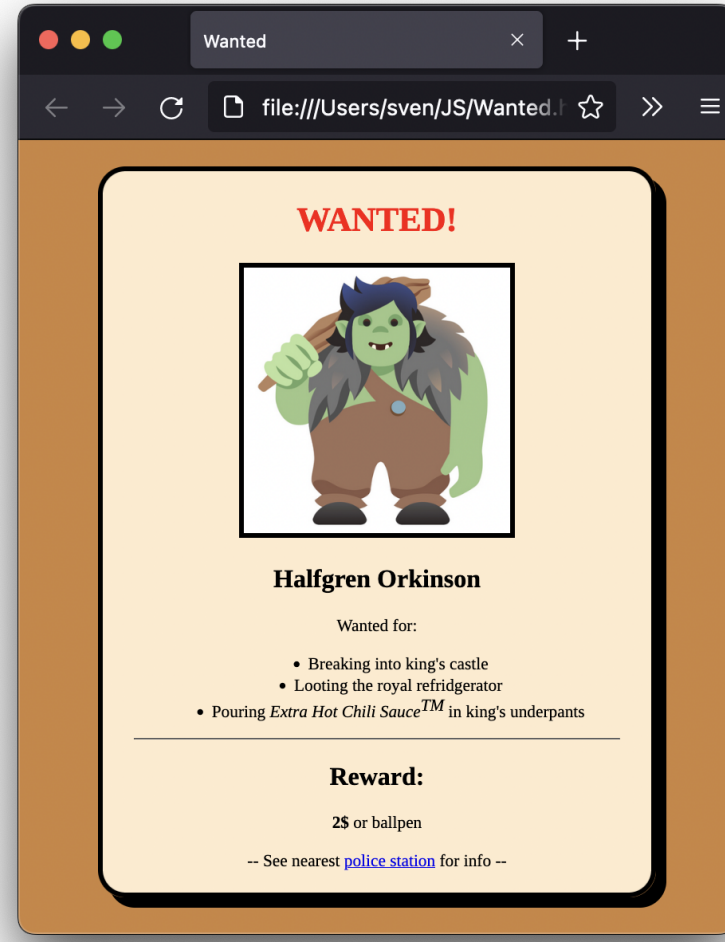


Figure 38. A poster straight out of some cheesy movi

Here comes the HTML code, but before you read it you should first try to create such a *"Wanted!"* poster with your own so far acquired coding knowledge! After that you can always look at the example code here in the book and see one (more) way how such a page was created.

```
<!DOCTYPE html>
```

```
<html style="background-color: Peru;">
    <head>
        <meta charset="utf-8">
        <title>Wanted</title>
    </head>
    <body style="background-color: Blanchedalmond; text-
align: center; width: 75%; margin: 25px auto; padding:
5px; border: 5px solid Black; box-shadow: 10px 10px;
border-radius: 25px;">

        <h1 style="color: Red;">WANTED!</h1>
        <img src="images/orc.jpg" width="250px"
border="5px" alt="Photo">
        <h2>Halfgren Orkinson</h2>
        <p>Wanted for:</p>
        <ul>
            <li>Breaking into king's castle</li>
            <li>Looting the royal refridgerator</li>
            <li>Pouring <i>Extra Hot Chili Sauce<sup>
TM</sup></i> in king's underpants</li>
        </ul>
        <hr width = 90%>
        <h2>Reward:</h2>
        <p><b>2$</b> or ballpen</p>
        <p>-- See nearest <a
href="https://en.wikipedia.org/wiki/Keystone_Cops">police
station</a> for info --</p>
  </body>
</html>
```

Sharing & uploading your HTML files

You now have a pretty solid basic knowledge of how to create appealing HTML pages. If you want to show your HTML pages to other people, you could of course show these on your own computer. Another option would be to give other people a copy of your HTML files. There are quite a few ways to do that:

You could just copy your files to a USB memory stick or any other type of storage media and hand it over to other people who then copy your files over to their devices. Or you could use the Internet to send your HTML files to other people, for example as an email attachement or using some sort of online file hosting service. No matter which way you choose, there are some things you should bear in mind:

If you only have a single HTML page without images that you want to pass on, then you can simply copy and pass on just this one file. However, if you use images or other files in your HTML page, then you must somehow pass on these images together with the HTML page. And you have to share the images in the same folder hierarchy as you have them on your computer.

What sounds quite complicated is actually quite simple and logical: Let's assume, for example, that you have saved your HTML page in a folder called "MyFiles". And you have saved your images in another folder called "MyPictures", which is located in the "MyFiles" folder. (You could also say that this folder "MyPictures" is a so-called *subfolder* of "MyFiles").

If you now pass on your HTML files, you must ensure that this folder hierarchy is retained. In our example, this means that the folder "MyPictures" remains a sub-folder of "MyFiles" in the copy that you want to pass on. This also means that you cannot save the

pictures anywhere else in the copy: When displaying your HTML file, the browser searches for the images at the exact location described in your HTML code. If the image folder or the images are now in a different location in the copy, the browser does not know where to find these images and will simply display an empty space instead of the images. So always save your files in the same folder hierarchy as you have them on your own computer.

The same also applies to your other files, for example downloadable PDF files or HTML files you have written that you link to each other (i.e. you write a link in one of your HTML files that leads to another of your HTML files): Here, too, you must then maintain the same folder hierarchy - otherwise clicking on a link in the copy will result in an error message from the browser that the linked file could not be found.

Your own homepage on the Internet

If you want to create your own homepage which should be accessible at any time and from anyone on the Internet, it basically works as described above for copying your HTML files. Now you just don't copy your files to a USB memory stick, but via the Internet to a computer of the website hosting company that is always switched on and connected to the Internet. (Sometimes such an always-on and always-Internet-connected computer is called a "server").

There are many different hosting offers: Some are free, but the storage space you can get may be rather small and you may have to allow advertising from the hosting company to be displayed on your homepage. Many hosting offers are fee-based and depending on the offer (and the price you have to pay...) you get a lot of storage space and other bonuses such as backup service, 24-hour help hotline, etc.

What all these hosting offers have in common is that you then

receive an Internet address at which the copies of your files uploaded to the server can be accessed by anyone. This Internet address is also called an *URL* ("Uniform Resource Locator"). For example, the URL of this book is `http://sven.kir.jp/JS`.

Now anyone who enters this URL in their browser can see your files on the Internet.

It has become common practice to name the first page or the starting page (also known as "home") `index.html`.

Further reading

You have finished reading this book but still can't get enough of HTML? Don't worry, we've got you covered! Here are some great resources on the topic of further HTML programming.

Any literature listed here is free to download from the Internet and is released under a similar free license as this book.

In the case that a link listed here is no longer accessible, it would be an idea to enter the book title into a search engine and look for a new working link.

HTML

Mozilla Developer Network

Not a "book" in the real sense, but rather an online reference work, offering everything you could want on the subject of HTML. Created by the makers of the Firefox web browser:

https://developer.mozilla.org/en-US/docs/Learn/HTML

HTML5 Notes for Professionals

A collection of guides and information compiled from *Stack Overflow*:

https://goalkicker.com/HTML5Book/

Learn JavaScript the fun way - Programming basics for beginners

Figure 39. The next book in this series

The direct successor to this book teaches you how to add interactivity to so far static HTML websites using *JavaScript*. From a simple calculator app, to a dog-year converter, to a math trainer or even a rocket launch game, you will be introduced to app programming in simple steps.

And just as with this book, all the apps presented can be run on any device with a reasonably modern web browser without any customization, and nothing more than a standard text editor is required to create them. Of course, this book is free and freely licensed just like the book you are reading right now:

http://sven.kir.jp/JS

License

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

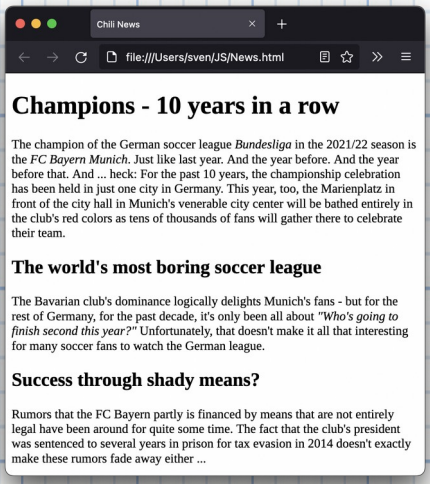
No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

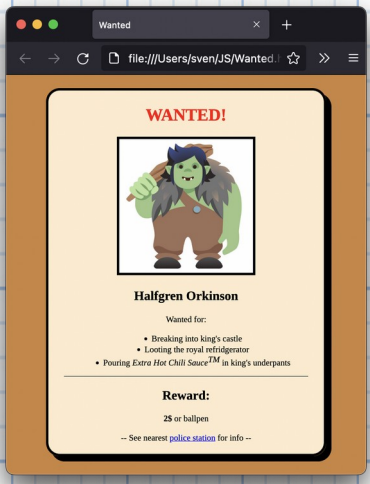
Want to try your hand at creating your own *HTML* webpage with your computer, but don't have any programming experience? Then this is the book for you!

Absolute programming beginners will be taught in easy-to-understand English to create their own funky websites that will run on computers, tablets, smartphones ... basically they run on any device with a browser!

For example create a webpage that looks like a proper newspaper, or another one that resembles a *"WANTED!"* poster from some cheesy movie.

This is the first book in the ***Programming basics for beginners*** series and is followed by ***Learn JavaScript the fun way***.





Sven Koerber-Abe is an associate professor at the Faculty of Science and Engineering of the Aoyama Gakuin University in Tokyo.

http://sven.kir.jp/JS