How to Code
in

HTML5

and

CSS3

Damian Wielgosik

Foreword

We live in a time when websites have become part of our everyday lives, replacing newspapers and books, and offering users a whole range of new opportunities. You probably visit at least a few of your favorite places on the Web every day, whether it's for shopping, sending messages, playing games, checking the news or looking at pictures of your friends. With the help of websites, you can have fun, make a living, and even get to know other people.

So it seems that the ability to create websites would be extremely valuable. The Internet provides us with many opportunities for development, and knowledge about how the Internet is built allows us to understand the changes taking place in modern society and economy.

But have you ever wondered how to create your own website?

If you have, I invite you to take a journey with me. We will look at websites that you know and visit every day. By analyzing them, we'll use comparisons and analogies that will help you better understand how they are built. I've been a web developer and working on the web professionally for several years and I am confident that at the end of our adventure together, we will create your first website.

Websites and Legos

I really like The Verge. It's a website where you can find interesting articles about new technologies, science and culture. On the home page, you can find dozens of feature stories and articles which tend to look similar to each other. They have a title, category, date, and image entry. It almost looks like it was built with bricks.

Let take a look at one of them:

At first glance, there's nothing complicated. Let's have some fun and highlight (or block out) each element, as though we were dealing with Lego parts.

Back to School 2013: The Verge buying guide

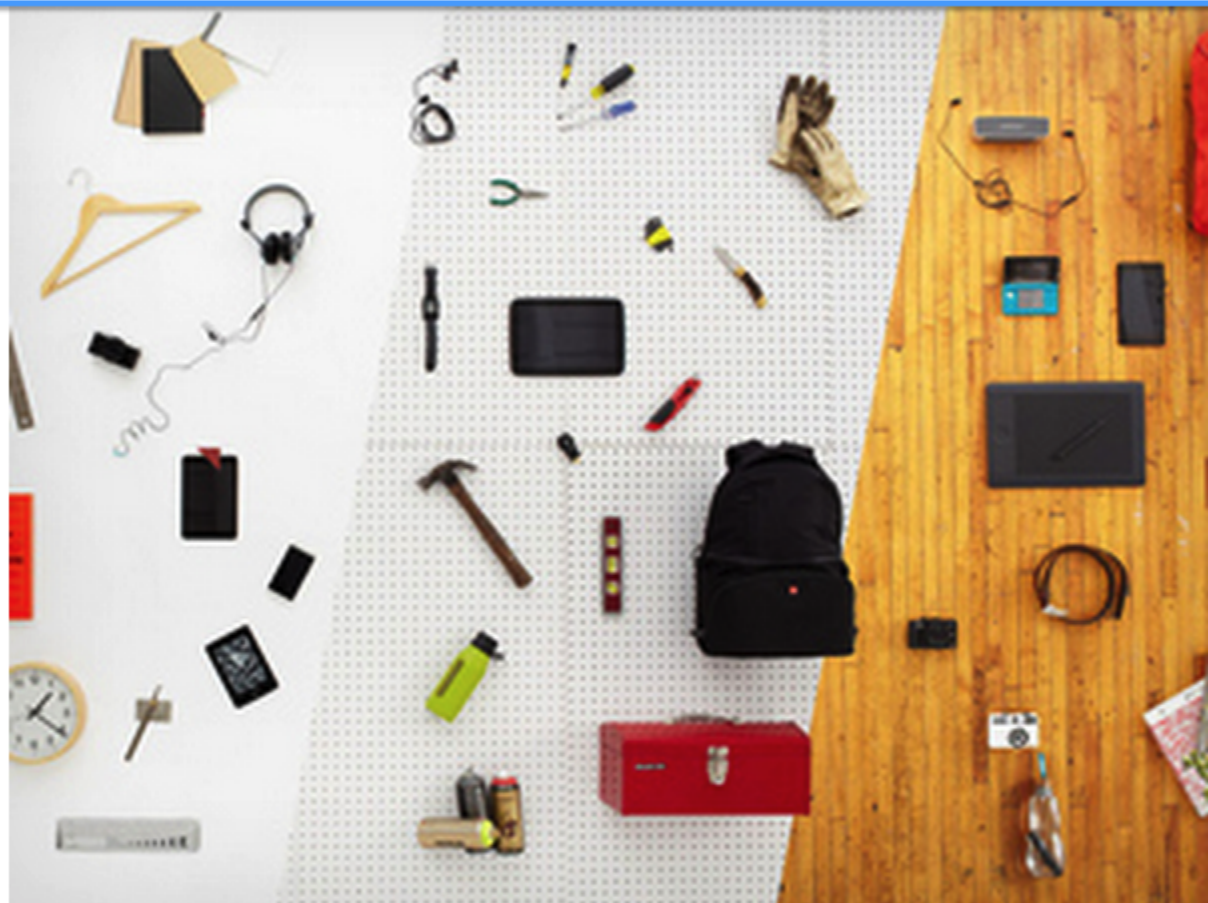**Verge Staff** | August 22, 2013 11:02 am

First, the bad news: summer's over. It's chillier out there every day, and soon you won't have beaches, road trips, or *kaiju* to entertain you. It's time to pull your jacket out of the closet and get to work.

Now the good news: You can make your own fun, you just need the right gear. So...

In total, we've got five pieces, arranged one on top of the other.

You might remember from your childhood that in order to build something specific, you need a lot of different blocks that are useful for different things. Each of them has a certain function. For example, when building a house, one type is useful for walls, and the other more useful for floors. There is no single, universal block or element from which you can make anything that comes to mind. See the figure below? This is what a large selection of lego choices should look like, right?

The same is true with websites. When you build a website, you use the different elements according to their ideal destination. In the Verge example, it seems that we

are dealing with various objects (or blocks), so doing something like applying the same color or style to each block would be counterintuitive. After all, the title is not the same as the date, or content of the paragraph. They each perform completely different functions.

In order to continue then, we need to identify elements of the article by the functions they perform on the site. Let's do this by adding a unique color to each "block."

This is much more interesting. We now have a few different types of "building blocks". Only two blocks are of the same type, specifically the two paragraphs below

the picture. This is no different than organizing legos. We're going to keep together similar fragments of text that belong in the same group. As a formality, we'll name each of the sections based on their function in the context of the article.

So we've marked each element according to their semantic meaning. This is exactly the kind of behavior and logic that we can expect to see from a web browser. It's our job to tell the browser, in a way that it can understand, what each of these elements

mean and how they fit together semantically. If this is not done, then our site will appear as a clump of single text.

You might have created such pages or articles using a text editor program like Microsoft Word or Pages. In text editors, achieving effects like "header styles" is a matter of clicking a few buttons. In other words, when we select text in a word editor and press "Header 1" we are assigning a bunch of different features in the background that tell the editor to display stuff in a specific way.

Therefore, if we wanted to recreate a page like the above example in a word processor, it would be simple and easy, at which point you could probably close this book and go do something else (Breaking Bad would be a good alternative). The problem is that we want to display this article on the web, which must be displayed in a browser, and never in a text editing program.

Let's build our first website

Let's suppose that we want to build a webpage with a job posting. It should look something like this:

We're looking for an HTML and CSS developer

For our client, The Cat Factory, **we need a skilled web developer in HTML and CSS**. We offer a competitive salary, a bag of cat food, and toys.

Don't wait, apply now! Our crazy team is waiting for you right meow!

Before creating any kind of web page, it's a good idea to divide the content into smaller components by their importance. Let's now try to identify and highlight each element of this job posting, just as we did in the example from The Verge.

We're looking for an HTML and CSS developer

For our client, The Cat Factory, **we need a skilled web developer in HTML and CSS**. We offer a competitive salary, a bag of cat food, and toys.

Don't wait, apply now! Our crazy team is waiting for you right meow!

A brief analysis will help us to better understand which areas of the text should stand out in the job posting. Red indicates the headline text. Green indicates the accompanying image. And purple marks the two paragraphs (or "body") of the job posting.

Let's return for a moment to the analogy of word processors and text editing programs where web pages could be created in regular text documents. Perhaps Open, Notepad, or Word, would contain the text of the announcement, and then save as a text file. It should work right?

```
jobpost.txt
1   We're looking for an HTML and CSS developer¬
2   ¬
3   [image]¬
4   ¬
5   For our client, The Cat Factory, we need a skilled web developer in HTML and CSS. We offer
.   a competitive salary, a bag of cat food, and toys.¬
6   ¬
7   Don't wait, apply now! Our crazy team is waiting for you right meow!
```

If you check this in a web browser, you get the following:

```
jobpost.txt                    ×

file://localhost/Users/ferr/Documents/jobpost.txt

We're looking for an HTML and CSS developer

[image]

For our client, The Cat Factory, we need a skilled web developer in HTML and CSS. We
offer a competitive salary, a bag of cat food, and toys.

Don't wait, apply now! Our crazy team is waiting for you right meow!
```
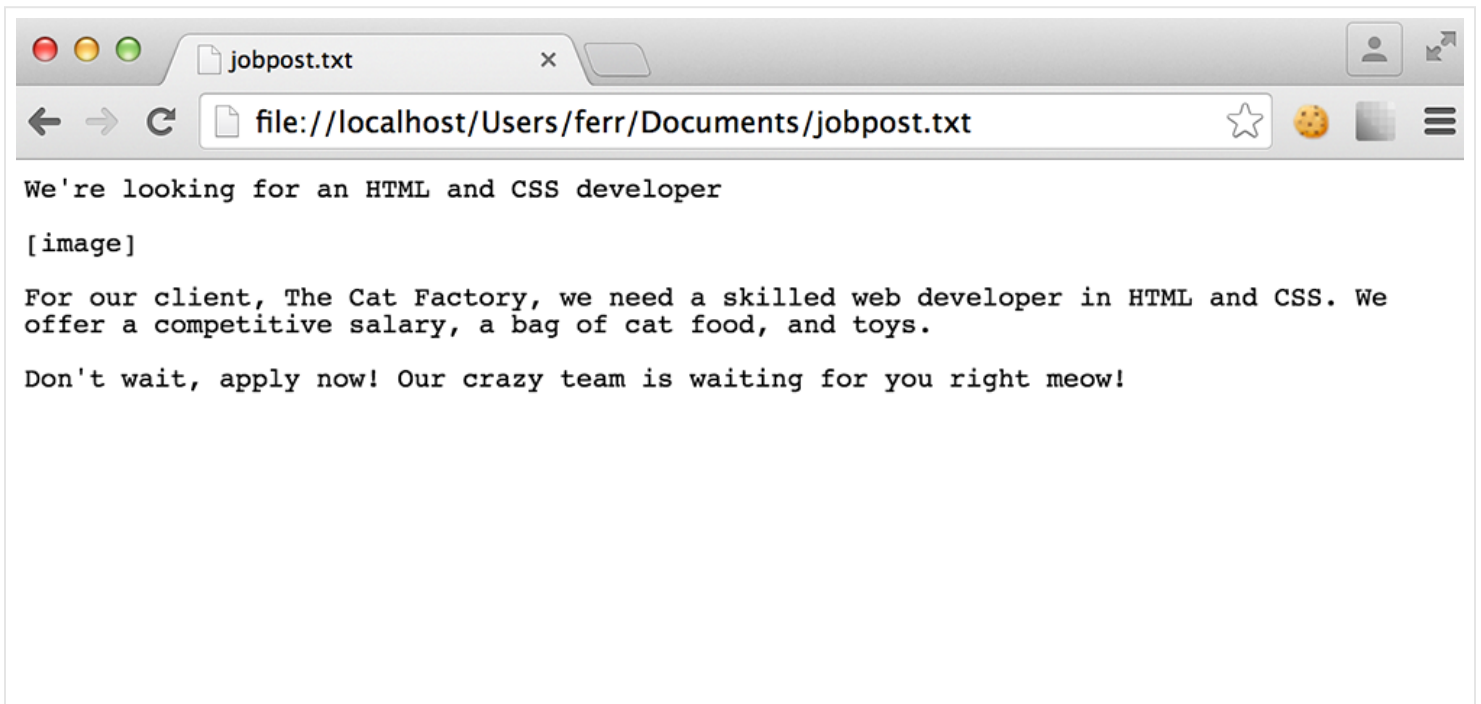
However, this doesn't look like what we designed, does it? It's just a mass of text and doesn't display a picture. What now? Maybe we should try to create this post in Word or Photoshop? We don't exactly want to do this. We've made an error here that we'll soon fix. The key problem is that we've created a website with only plain text.

A web browser cannot understand how to display a page properly with only plain text — it doesn't know which part of the text should be the title or which part should be a picture. In order to display the page properly, we need to define each element by the function of the text and pass this information to the browser. We've partly done it. Let's do a little definition work and use proper syntax so browser will also understand it:

As you can see, there are special markings within the text. We'll get to these later. For now, it's best to simply try to copy them exactly as they are, from top to bottom, as if you were building with blocks, one by one.

You should get something like the following:

```html
<h1>We're looking for an HTML and CSS developer</h1>

<img src="images/white-cat.jpg">

<p>For our client, The Cat Factory, we need a skilled web developer in HTML and CSS. We offer a competitive salary, a bag of cat food, and toys.</p>

<p>Don't wait, apply now! Our crazy team is waiting for you right meow!</p>
```

Next, save the HTML file. By the way: I recommend you Sublime Text Editor which is one of the best code editors out there.

Now let's try the new file in a web browser.

We're looking for an HTML and CSS developer

For our client, The Cat Factory, we need a skilled web developer in HTML and CSS. We offer a competitive salary, a bag of cat food, and toys.

Don't wait, apply now! Our crazy team is waiting for you right meow!

It looks a bit better. Each section of the text is now formatted differently, and the picture is also displayed. This is not the final version though. The lack of color, bold text and style doesn't quite match up yet. But before we work on that, let's keep our focus on the structure of the page.

First, let's compare the plain text document version of the posting, and the one presented in HTML:

```
  1  <h1>We're looking for an HTML and CSS
     developer</h1>
  2
  3  <img src="images/white-cat.jpg">
  4
  5  <p>For our client, The Cat Factory, we
     need a skilled web developer in HTML
     and CSS. We offer a competitive salary,
     a bag of cat food, and toys.</p>
  6
  7  <p>Don't wait, apply now! Our crazy
     team is waiting for you right meow!</p>
```
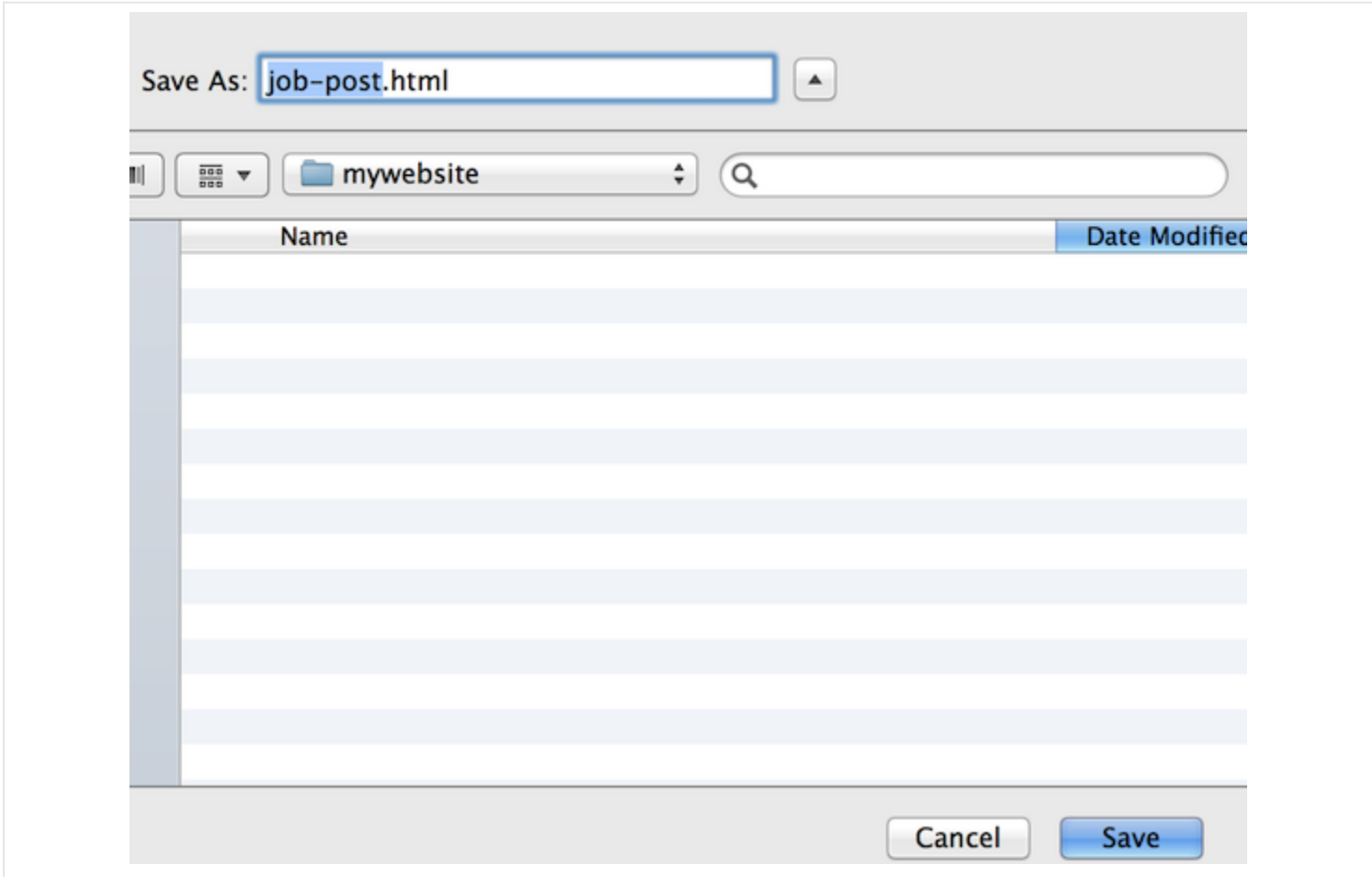
```
  1  We're looking for an HTML and CSS
     developer
  2
  3  [image]
  4
  5  For our client, The Cat Factory, we
     need a skilled web developer in HTML
     and CSS. We offer a competitive
     salary, a bag of cat food, and toys.
  6
  7  Don't wait, apply now! Our crazy team
     is waiting for you right meow!
```

As a result of surrounding specific text fragments with special tags, we've created a completely different document, which can be understood by the browsers. Way to go!

This is called HTML, or Hypertext Markup Language, and is the primary markup language for displaying information for a web browser. In simpler terms, it's a language that uses "tags" (like <this>) to mark text, so that you can describe the text to your browser. This also includes search engines and a few other programs which we'll cover later on. HTML tags are like grammar forms in verbal communication, without them, you cannot build a coherent website, much like you cannot form correct sentences or paragraphs without grammar.

In our case, we used only a few simple tags, and below is a list of the functions that each tag performs:

- <h1> - headline (of the first degree)
- <img> - picture element
- <p> - a paragraph of text

So, if you have a website with a few paragraphs of text, you surround each paragraph with the <p> tag. If the text is the main headline, then surround the text with <h1>, and so on and so forth.

All tag constructions are extremely simple and look like this:

```
<tagname>content</tagname>
```

You might have noticed that the second tag is different. This is because each tag must "close," as the saying goes in HTML jargon. The forward-slash appears in the closing tag to tell the browser that this marks the end of an element. For example, all paragraph elements end in </p>, while a Header 1 ends in </h1>.

There are also tags which do not need to be enclosed. For example image tags, <img>, which are used to insert pictures. In our example above, we used it as follows:

```
<img src="images/white-cat.jpg">
```

There are several more tags that do not need to to be "closed", which you will learn about later in this book.

One important realization to make about HTML tags is that, like dictionaries contain the vocabulary of a language, so too does HTML have a kind of dictionary that defines tags and describes when and where to use them. At the moment we have covered only a few, but there are many more. The international organization known as The World Wide Web Consortium (W3C) encourages best practices, standards, and compatibility across the web. You can view their definition of HTML elements here: http://www.w3.org/TR/html-markup/elements.html.

Also, it's helpful to remember that, like natural languages, HTML evolves over time. Today, we use the fifth version of the HTML language, known as HTML5.

Returning to our example, it should look like the following in HTML code:

```
<h1>We're looking for an HTML and CSS developer</h1>

<img src="images/white-cat.jpg">

<p>For our client, The Cat Factory, we need a skilled web
developer in HTML and CSS. We offer a competitive salary, a bag
of cat food, and toys.</p>

<p>Don't wait, apply now! Our crazy team is waiting for you
rightmeow!</p>
```

Notice that it contains two paragraphs of text, marked within <p></p> tags:

```
<p>For our client, The Cat Factory, we need a skilled web
developer in HTML and CSS. We offer a competitive salary, a bag
of cat food, and toys.</p>

<p>Don't wait, apply now! Our crazy team is waiting for you
rightmeow!</p>
```

Suppose that we want the reader of our ad to pay particular attention to a certain part in a paragraph. Let's say that we wanted to mark the part of the text that says "we need a skilled web developer in HTML and CSS." How would an HTML tag let the browser know? The answer is the <strong> element, which will surround parts of text that should represent a strong importance for its contents.

```
<p>For our client, The Cat Factory, <strong>we need a skilled
web developer
in HTML and CSS</strong>. We offer a competitive salary, a bag
of cat food,
and toys.</p>

<p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
```

Note that the <strong> tag surrounds the text, and also sits within the <p> tag. In programming terms, we would say that the <strong> tag is a "child" element within <p> because it nests within the parent. There are many tags that can nest within other tags, while many others cannot. Each tag has a specific list of possible HTML elements which it can contain (or be contained by), and you will need to check whether a certain tag is allowed or not. You can think of it like smaller blocks combining to make a bigger block piece (in this case the <strong> block is a component of <p>, which is in turn a block of the whole text).

Here is what it looks like on the web.

For our client, The Cat Factory, **we need a skilled web developer in HTML and CSS.**

Take note that a web browser we used above displayed the fragment in bold, but this is not always a rule. It's very common that <strong> makes text bold, but sometimes it will be ignored. The reason for using this HTML tag in this text is a good example of how nesting works, but visual effects (like bold) can be better achieved through a language called CSS, which we will learn more about in a moment.

Features of Perfect HTML Code

At this point we've established our first website. Unfortunately, according to the current standards of good web pages, we are not yet complete. Just as engineers are held to strict design guidelines, or restaurants have health and safety regulations, so too do web developers have a top set of rules that define a job well done.

Let's begin with the code we have so far, with all of the elements we've learned about:

```
<h1>We're looking for an HTML and CSS developer.</h1>
<img src="images/white-cat.jpg">
<p>For our client, The Cat Factory, <strong>we need a skilled
web developer in HTML and CSS</strong>. We offer a competitive
salary, a bag of cat food, and toys.</p>
<p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
```

At the very top of any self-respecting HTML page, there should be an indication of the document type. This is the doctype:

```
<!DOCTYPE html>
<h1>We're looking for an HTML and CSS developer.</h1>
<img src="images/white-cat.jpg">
<p>For our client, The Cat Factory, <strong>we need a skilled
web developer in HTML and CSS</strong>. We offer a competitive
salary, a bag of cat food, and toys.</p>
<p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
```

The highlighted section above "tells" the browser that the page is an HTML document, and the page's code should be treated according to the established standards for the language. This informs the browser of several rules, which will guide how our website is viewed. Among other things, it helps the browser to know which tags may be allowed or not allowed. It helps the browser better interpret our code.

Below the doctype, we want to put the marker <html>:

```
<!DOCTYPE html>
<html>
  <h1>We're looking for an HTML and CSS developer.</h1>
  <img src="images/white-cat.jpg">
  <p>For our client, The Cat Factory, <strong>we need a skilled
web developer in HTML and CSS</strong>. We offer a competitive
salary, a bag of cat food, and toys.</p>
  <p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
</html>
```

Note that this tag needs to be closed, so we want to be sure to add </html> at the end:

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

As a result, everything that belongs to our website is contained under the tag <html>. When writing HTML, it only makes sense that the code should be enclosed by </html>.

However, the <html> tag is not yet complete. We want to add more information about the language of our site. Since this is an English example, let's designate it as such:

```
<!DOCTYPE html>
<html lang="en">
  <h1>We're looking for an HTML and CSS developer.</h1>
  <img src="images/white-cat.jpg">
  <p>For our client, The Cat Factory, <strong>we need a skilled
web developer in HTML and CSS</strong>. We offer a competitive
salary, a bag of cat food, and toys.</p>
  <p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
</html>
```

Notice the highlighted section has modified the <html> tag. This is called an "attribute." Attributes are modifiers in HTML, and are always written next to the tag, between the <> enclosure. Attributes have the following template when writing code.

```
tag attribute="value"
```

As a HTML developer grows, he/she will come across many different attributes. In addition to "lang" there are a large variety. Here are just a few examples.

- <div id="sidebar"></div> - the name of the attribute is "id", and the value is "sidebar"
- <p class="landscape"></p> - the name of the attribute is "class" and the value is "landscape"
- <input type="text"> - the name of the attribute is "type", and the value is "text"

We use attributes because in many HTML tags (for example <html>) do not contain all the necessary information. With attributes, we can modify the tags we use and add even more useful information to them. In this case, we have let the browser

known that our HTML document is written in English, so we've modified the <html> tag with the attribute "lang" and given it a value for "en" (English).

Multiple attributes can even occur in a single tag. Here are a few examples:

```
<input type="text" value="enter text here">
<a href="http://functionite.com" title="We Train
Developers">HTML Training</a>
```

It's important to note that attributes are also part of HTML standards, just like nesting tags, and lists of these attributes, along with the tags that they can modify, can be found on the web.

In fact, we've already used an attribute when we were first building our web page! Remember the image code? You placed it in an HTML document with the following line:

```
<img src="images/white-cat.jpg">
```

In this case, the <img> tag was modified with the attribute src with the value of images/white-cat.jpg.

The tag by itself is only <img>, so if we simply left it alone, the browser would have no way to retrieve the source of information to display. When we define a "src," we tell the browser, "Hey, load information from this source." The browser then knows it should look for the folder "images" and the file "white-cat.jpg." We can also pass a link to any resource on the web (if it's not restricted), for instance http://4.bp.blogspot.com/-z4sMggeD4dg/UO2TB9INuFI/AAAAAAAAdwc/Kg5dqlKKHrQ/s1600/funny-cat-pictures-032-025.jpg.

This is the gist of how attributes work. Thanks to attributes, we can add rich and valuable information that is helpful for HTML tags.

Now let's return to our document and add in more necessary components. We'll be looking at the tag <head>, which contains elements that are not necessarily helpful to your page's content, but essential for search engines, browser tabs, and so on. For example, the title on a web page helps search engines find your content.

```
<!DOCTYPE html>
<html lang="en">
   <head>
   </head>
   <h1>We're looking for an HTML and CSS developer.</h1>
   <img src="images/white-cat.jpg">
   <p>For our client, The Cat Factory, <strong>we need a skilled
web developer in HTML and CSS</strong>. We offer a competitive
salary, a bag of cat food, and toys.</p>
   <p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
</html>
```

At the moment, our head tag is empty, but usually this is the location you want to put all meta information like the title of the whole document. As we create the page with our job offer, let's say that we decide on a title "Job Offer: HTML and CSS developer." Let's go ahead an add it!

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Job Offer: HTML and CSS Developer</title>
  </head>
  <h1>We're looking for an HTML and CSS developer.</h1>
  <img src="images/white-cat.jpg">
  <p>For our client, The Cat Factory, <strong>we need a skilled
web developer in HTML and CSS</strong>. We offer a competitive
salary, a bag of cat food, and toys.</p>
  <p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
</html>
```

As you might have noticed, we added the title between the tag <title>. What's the difference, and why did we do it that way? The below image explains what happens where we don't have a <title> tag (left), and our page that has the <title> tag (right). With title, our websites are more usable for browser user and navigating among many tabs is much easier.
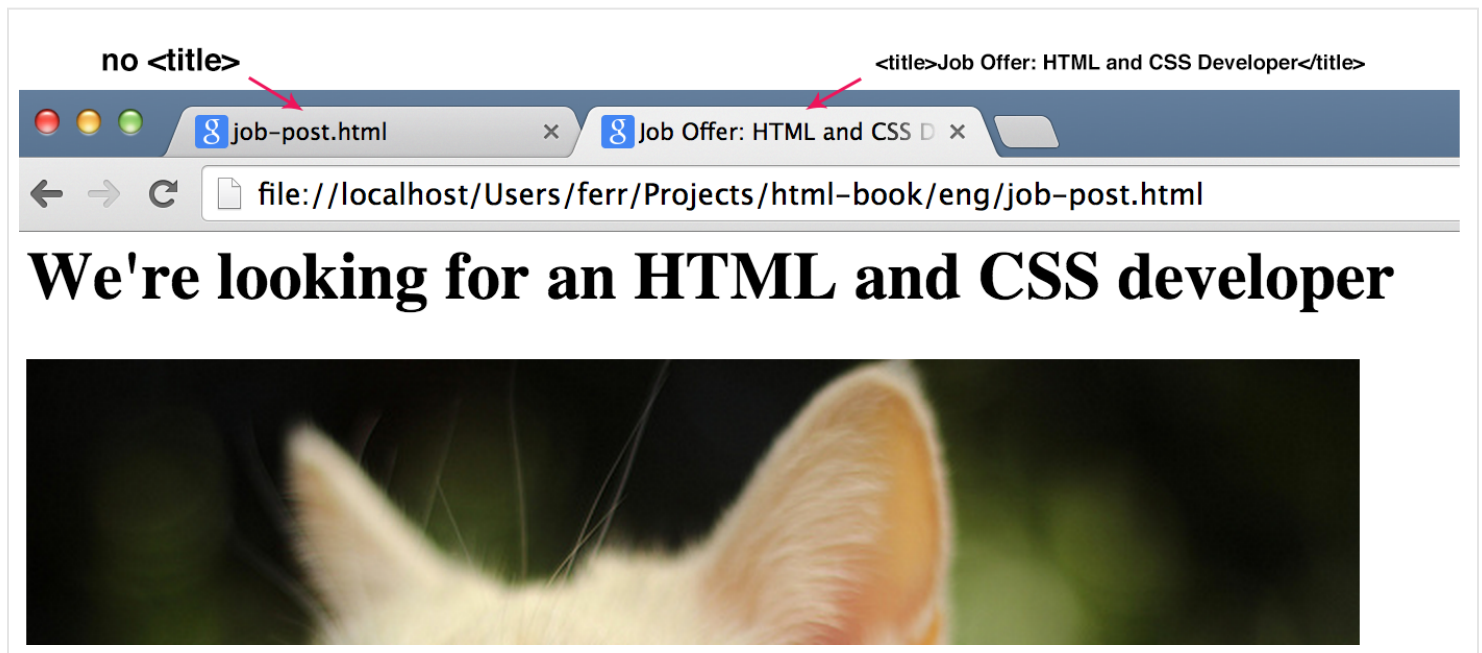
Another example might be my company website. I've used "Functionite - JavaScript workshop team" as a text inside title tag. Thus, it appears as that in Google search results:
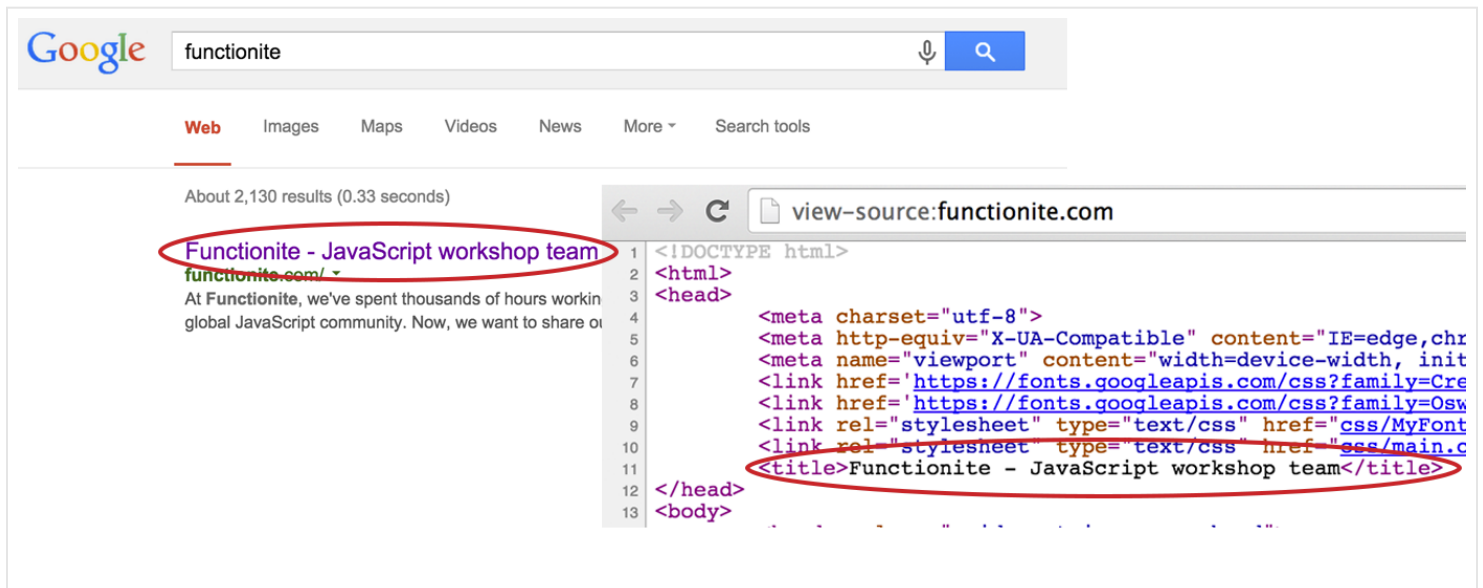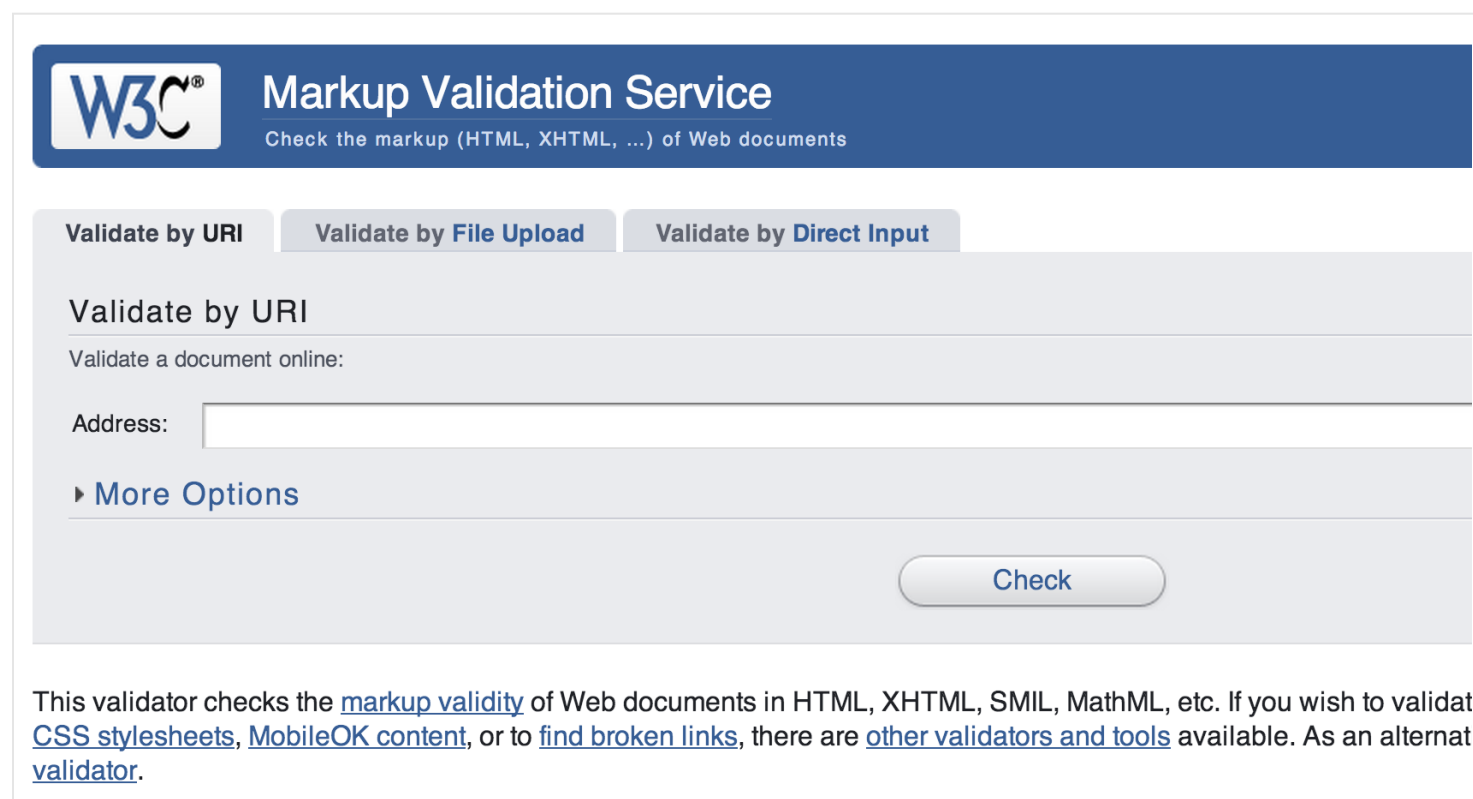
As you can see, this is a very important to consider when coding your site. Without it, the site would be without a name, which would leave a bad impression on visitors. Personally, I wouldn't trust a page that doesn't even have a title.

We need to do one more thing after adding a head, and that is to add the <body> tag. This separates the content of the entire page from the head. Everything displayed on your web page will be placed within <body>.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Job Offer: HTML and CSS Developer</title>
  </head>
  <body>
    <h1>We're looking for an HTML and CSS developer.</h1>
    <img src="images/white-cat.jpg">
    <p>For our client, The Cat Factory, <strong>we need a
skilled web developer in HTML and CSS</strong>. We offer a
competitive salary, a bag of cat food, and toys.</p>
    <p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
  </body>
</html>
```

Hooray! It looks like the end of our efforts and it seems that everything is correct. As a novice coder, it's probably a good idea to check if everything is OK (e.g. nested tags, good attributes, etc.). But not everyone has friends among advanced HTML developers, who may be asked for consultation. Luckily, there is a special tool we can use to check if our code is correct. The W3C organization, the same group that defines the standards and upholds best practices of HTML documents, has made a tool to check code. The tool is simply called an HTML validator, and is available at http://validator.w3.org.

The page should look like the one below:

Let's use the tool and check to see if our code is correct.

...Woops! We have one error and 3 warnings. Let's take a look at them.

The single error that the validator reported has to do with our image. Let's take a look at the content of the error:

> An img element must have an alt attribute, except under certain conditions. For details, consult guidance on providing text alternatives for images.

It turns out that the <img> tag in your document must always have the "alt" attribute that will be used in case the browser cannot load the image. It will then display the value of this alternate attribute instead (it's better than an empty space, right?).

Before we choose the value for the attribute "alt," let's look at the HTML specification to make sure that we do it correctly:

Here is the page where we can check it: W3C:

> "Gives the fallback content for the image. The requirements on the alt attribute's value are described in the next section."

It then further distinguishes several scenarios, for when and how to use the alt attribute. In our case, we are dealing with an image that has been placed for decoration of the post and does not contain any content, plot, or other information to the reader. In this case, we should leave the alternate attribute empty, says the specification:

In other cases, we would want to choose a text string that might replace what is on the picture. Detailed guidelines for several distinguished cases can be found at the specification for the <img> tag here: http://www.w3.org/wiki/HTML/Elements/img.

Ok, so we know that we want to have the "alt" attribute empty.

Let's add it...

```
<img src="images/white-cat.jpg" alt="">
```

...And then check our document against the validator again.

	This document was successfully checked as HTML5!
Result:	Passed, 3 warning(s)

Source :	`<!DOCTYPE html>`
	`<html lang="en">`
	`    <head>`
	`        <title>Job Offer: HTML and CSS Developer</title>`
	`    </head>`
	`    <body>`
	`        <h1>We're looking for an HTML and CSS developer.</h1>`
	`        <img src="images/white-cat.jpg" alt="">`
	`        <p>For our client, The Cat Factory, <strong>we need a skilled web developer in HTML and CSS</strong>.`
	`We offer a competitive salary, a bag of cat food, and toys.</p>`
	`        <p>Don't wait, apply now! Our crazy team is waiting for you right meow!</p>`
	`    </body>`

It passed with no errors!

But we still have the 3 warnings. The most important of them is as follows:

But not to worry, this is a simple warning. Before we try to understand it, let's examine the following example which will allow us to better understand the problem.

It's important to realize that your website may be visited by people from different countries like Spain, China or Sweden not to mention others. If you allow it, they can leave a comment or a message that might be displayed next to your articles. It's important to note though, that different countries use different languages with their own diacritical marks. In Swedish it can be ä or ö, while Chinese offer you a whole range of them like 汉语拼音方案. In the global web it's crucial to let different languages display correctly on the same page. Let's examine what would happen, if we used Chinese language in our post:

```
<p>PS 再见</p>
```

Let's check the effect in the browser:

We're looking for an HTML and CSS developer

For our client, The Cat Factory, **we need a skilled web developer in HTML and CSS**. We offer a competitive salary, a bag of cat food, and toys.

Don't wait, apply now! Our crazy team is waiting for you right meow!

PS å†è‹‹

Note the last line where, instead of a proper Chinese sentence, we can see some odd signs that don't make sense at all in any of modern languages. Don't panic, it's related to the warning that we have received in the HTML validator. It turns out that we haven't inserted a special line, that would allow other languages to display properly. In computers world, it's called Character encoding. If we passed this information to the HTML page, it would "tell" the browser that it's ready to display a wide range of characters including all different diacritical marks.

Let's do this!

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Job Offer: HTML and CSS Developer</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>We're looking for an HTML and CSS developer.</h1>
    <img src="images/white-cat.jpg">
    <p>For our client, The Cat Factory, <strong>we need a
skilled web developer in HTML and CSS</strong>. We offer a
competitive salary, a bag of cat food, and toys.</p>
    <p>Don't wait, apply now! Our crazy team is waiting for you
right meow!</p>
  </body>
</html>
```

This time we have had to use a special tag <meta> with the attribute charset and given it a value for "utf-8". utf-8 means a special character set that incorporating those from a majority of languages around the world, if not all of them.

And that's it! You have just built your first website ever! Congratulations!

HTML Exercise: Coding a Blogpost

Time for another exercise that will show you the philosophy of writing more advanced websites in HTML5. We'll try to produce the following example as a website, an article on a blog.

Justin Beaver: Ever since I learned HTML, my life has made a complete 180

Posted by: Damian Wielgosik

Justin Beaver confessed something that even his greatest fans would have never expected of the skilled musicians and lyricist. The young rock-and-roller admitted that since he typed his first title tag, his life became easier. It has been reported by those surrounding the Canadian that Beaver's private mentors, Ryan Loseling and Nicolas Crate, often walk around Los Angeles disputing what a great tool the HTML validator is.

Justin Beaver's happy cat

Beaver has already created some websites and does not intend to stop there. "I will probably have a song about HTML on my next album," - the artist added.

Note that here we have one header for the article. In addition, information about the author, as well as a photo and a "quote" paragraph. It is worth noting that as a front-end web developer, you will often receive graphic designs (like the one shown above) and have to recreate everything in HTML.

Let's begin with the elements which will always be fixed and do not change. As we did earlier, we want the doctype tag, <html>, the <head> and <body>. Note that in

our <head> we'll want to make sure we have character encoding for "utf-8" so that we can display special characters on our site.

Our starting template for the code will look like this:

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

Let's add the title now:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Justin Beaver fascinated by HTML</title>
  </head>
  <body>
  </body>
</html>
```

At this point, we've finished the template which will now serve as the basis for the contents of the article.

If we save the file as article.html, and then open in a browser, we will see nothing at all because the <body> tag has no content.

We generally start from the most general content, followed by more specific, detailed content as we progress. Let's analyze our pieces:

1. In the article, the most important information would probably be the header information which contains the title: "Justin Beaver: Ever since I learned HTML, my life has made a complete 180."

2. The next most important information is about the author, "authored by Damian Wielgosik."

3. The first paragraph of text comes next, then a place for a photo and its description.

4. Finally, the last paragraph, which contains a quote, "I will probably have a song about HTML on my next album".

And those are our pieces. This type of analysis where we look at the pieces helps us visualize how the HTML code will look. The hierarchy starts from the most general element (the parent), and then continues to elements which a parent contains (children), which are more specific, detailed elements.

With this understanding of hierarchy, our information should be organized as follows in the HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Justin Beaver fascinated by HTML</title>
  </head>
  <body>
    <article>
    </article>
  </body>
</html>
```

You might have noticed that we used the <article> element which is contained within the <body> parent. Articles consist of a header and content, so all tags representing heading and paragraphs will naturally be children of <article>.

Adding to our list of elements that we've discovered so far, we're going to now use an element that marks headers, the <header> tag.

We can add the <header> code as a child of <article>. As we move from top to bottom in the code (in terms of importance), note that we also move rightwards with indentations to show that an element is lower on the hierarchy than its above component. In the example below, <header> is not only below <article>, but also indented to show where it is on the hierarchy.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Justin Beaver fascinated by HTML</title>
  </head>
  <body>
    <article>
      <header>
      </header>
    </article>
  </body>
</html>
```

Our header section is now ready, but what should go inside the <header>? Well, we discussed earlier that we have a title and author. We'll use the tag <h1> for the title of the text, and <p> for the author. There is no specific HTML tag for author, so in this case we're using <p> as a general container for a text.

In the code below, we've added <h1> and <p> tags to the header element:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Justin Beaver fascinated by HTML</title>
  </head>
  <body>
    <article>
      <header>
        <h1>Justin Beaver: Ever since I learned HTML, my life
has made a complete 180</h1>
        <p>Posted by: Damian Wielgosik</p>
      </header>
    </article>
  </body>
</html>
```

By the way, you should note that in addition to <h1>, there are lower levels of the HTML headers that look like this:

- <h2>

- <h3>

- <h4>

- <h5>

- <h6>

These elements help to map the logical structure of headings and subheadings. So for example, if we were writing in book format, its title would be contained between the element <h1>. Then, the names of the chapters will be tagged with <h2>, and subsections <h3>.

Note that you shouldn't simply denote a header as <h4> on a whim. Headers must be nested under a higher priority header, so if you have <h4>, <h3> must occur before it, and <h2> must occur before that, and so on.

For example, this book could look like this:

```
<h1>Simple HTML5</h1>
<h2>My first website</h2>
<h3>W3C Validator</h3>
<h2>Meet CSS</h2>
<h3>Selectors in CSS</h3>
```

Let's move on to coding the first paragraph of text. We want to avoid nesting the paragraph under the <header>. It makes sense that the paragraph should be part of the <article>. So, we'll add the first <p> within <article>, with the same priority as <header>, but underneath it:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Justin Beaver fascinated by HTML</title>
  </head>
  <body>
    <article>
      <header>
        <h1>Justin Beaver: Ever since I learned HTML, my life
has made a complete 180</h1>
        <p>Posted by: Damian Wielgosik</p>
      </header>
      <p>Justin Beaver confessed something that even his
greatest fans would have never expected of the skilled musicians
and lyricist. The young rock-and-roller admitted that since he
typed his first title tag, his life became easier. It has been
reported by those surrounding the Canadian that Beaver's private
mentors, Ryan Loseling and Nicolas Crate, often walk around Los
Angeles disputing what a great tool the HTML validator is.</p>
    </article>
  </body>
</html>
```

Next we'll add the photo plus a description. For this type of content, and therefore all related to the entire document such as photos, charts or maps, we'll use the `<figure>` tag. It is worth noting that the `<figure>` condition of use is important into that you can use the extra element `<figcaption>`, which puts a description of the image beneath it.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Justin Beaver fascinated by HTML</title>
  </head>
  <body>
    <article>
      <header>
        <h1>Justin Beaver: Ever since I learned HTML, my life
has made a complete 180</h1>
        <p>Posted by: Damian Wielgosik</p>
      </header>
      <p>Justin Beaver confessed something that even his
greatest fans would have never expected of the skilled musicians
and lyricist. The young rock-and-roller admitted that since he
typed his first title tag, his life became easier. It has been
reported by those surrounding the Canadian that Beaver's private
mentors, Ryan Loseling and Nicolas Crate, often walk around Los
Angeles disputing what a great tool the HTML validator is.</p>
      <figure>
        <img src="cat.jpg" alt="Justin Beaver's cat is pleased">
        <figcaption>Justin Beaver's happy cat</figcaption>
      </figure>
    </article>
  </body>
</html>
```

After we've added the image, we just have one more paragraph to add to the article. Note that this paragraph has a quote, "I will probably have a song about HTML on my next album." We can annotate this quote so that our code has a greater

semantic value. Perhaps in the future, someone will look for quotes from Justin Beaver, and this designation will help them find the quote faster. Otherwise, search engines would always have to deal with one huge chunk of text.

In order to indicate that a section of text is a quote, we're going to use the `<q>` tag within the new paragraph:

```
<!DOCTYPE html>
<html lang="en">
   <head>
     <meta charset="utf-8">
     <title>Justin Beaver fascinated by HTML</title>
   </head>
   <body>
     <article>
       <header>
         <h1>Justin Beaver: Ever since I learned HTML, my life
has made a complete 180</h1>
         <p>Posted by: Damian Wielgosik</p>
       </header>
       <p>Justin Beaver confessed something that even his
greatest fans would have never expected of the skilled musicians
and lyricist. The young rock-and-roller admitted that since he
typed his first title tag, his life became easier. It has been
reported by those surrounding the Canadian that Beaver's private
mentors, Ryan Loseling and Nicolas Crate, often walk around Los
Angeles disputing what a great tool the HTML validator is.</p>
       <figure>
         <img src="cat.jpg" alt="Justin Beaver's cat is pleased">
         <figcaption>Justin Beaver's happy cat</figcaption>
       </figure>
       <p>Beaver has already created some websites and does not
intend to stop there.
       <q>I will probably have a song about HTML on my next
album,</q> - the artist added.
       </p>
     </article>
   </body>
</html>
```

Now, let's save the code to a file with the extension .html and display it in the browser.

You have now completed your second page. Great! This is another step to becoming a professional web developer.

Site Visuals in CSS3

You may have noticed that, with exception of our images, our site does not look very interesting. Black text on a white background isn't very welcoming to our visitors. Let's work on the appearance of our site with CSS (Cascading Style Sheets). Using this language, we can manipulate traits of the site like color, font size, and many other qualities. There is a huge list of possibilities available with CSS.

In the previous sections, we used HTML to describe the content of the site, and divide it into fragments according to their importance. CSS will be responsible for the appearance of our web sites. CSS code can be placed in a separate file with the extension .css and inserted via a special HTML tag. You can also put it directly into the HTML document.

Imagine for a moment, in this abstract example, that we want to construct a house with CSS code, where we choose items such as windows, doors, roofing, walls, gutters and so on. We would want to buy windows of specific sizes, and paint for each of the necessary parts. If we built this house in CSS, one of the many solutions to this task might look like this:

```
roof {
    background-color: green;
}

doors {
    background-color: yellow;
    width: 100px;
    height: 300px;
}

windows {
    border: 5px solid brown;
    width: 150px;
}
```

Let's analyze the "roof" block of this code from top to bottom (note that top-to-bottom reading is a rule when reading all types of code, not just HTML and CSS).

```
roof {
    background-color: green;
}
```

If we translate the code above into normal English, we have chosen an element called "roof" and set the background color to green.

```
windows {
    border: 5px solid brown;
    width: 150px;
}
```

The code above says, "for all windows, set the following: a frame (border) with a width of 5 pixels (5px), marked by a continuous line (solid) color (brown). Also, the window itself should have a (width) of 150 pixels (150px).

You might have noticed a recurring pattern in the code. On the first line, we write the name of the element (termed "selector"), and then define the appearance of that element between bracket.

The template has the basic structure shown below:

```
selector {
   property_name: property:value;
}
```

This type of construction is a typical CSS rule. The rule consists in turn of a selector (everything before the first bracket) followed by a list of properties that you write between the brackets.

There are various way to specify exactly what we want to design. Let's say that we only wanted to specify design for windows on the ground floor. What then? We could write something like:

```
ground floor window {
   border: 5px solid brown;
   width: 150px;
}
```

The result is that only the selector has changed. Instead of "window {" we have specified "ground floor window {"

This code reads from left to right like "find the ground floor, and then find its window and set the following values." If we put a sub-selector under "ground floor windows" like:

```
adjacent wall windows {
```

Then we're telling the browser: "find the ground floor window, windows next to it, and fill them with the following values," and so on. If you remember the analogy in which we talked about nested HTML tags as the children and parents, this is the same concept, elements nested within other elements.

Unfortunately a browser can't quite build a house, but our example tells us how CSS works. This analogy is useful because as we code, we're not always able to see the changes. But we can think of paragraphs (<p>) as windows, and doors as the header (<h1>), etc.

It would look like this:

```
p {
}

h1 {
}
```

Let's apply what we've learned in this analogy to our example and use the same ideas to add a little color and life to the Justin Beaver article.

Recall that our code looks like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Justin Beaver fascinated by HTML</title>
  </head>
  <body>
    <article>
      <header>
        <h1>Justin Beaver: Ever since I learned HTML, my life
has made a complete 180</h1>
        <p>Posted by: Damian Wielgosik</p>
      </header>
      <p>Justin Beaver confessed something that even his
greatest fans would have never expected of the skilled musicians
and lyricist. The young rock-and-roller admitted that since he
typed his first title tag, his life became easier. It has been
reported by those surrounding the Canadian that Beaver's private
mentors, Ryan Loseling and Nicolas Crate, often walk around Los
Angeles disputing what a great tool the HTML validator is.</p>
      <figure>
        <img src="cat.jpg" alt="Justin Beaver's cat is pleased">
        <figcaption>Justin Beaver's happy cat</figcaption>
      </figure>
      <p>Beaver has already created some websites and does not
intend to stop there. <q>I will probably have a song about HTML
on my next album,</q> - the artist added.
      </p>
    </article>
  </body>
</html>
```

Recall the analogy of building a house. Instead of building doors, windows, etc., we are dealing with elements like <article>, <p>, <header>, <body>, <figcaption> and so on. These tags build the page and now CSS will help to give them style.

I've prepared a screenshot on the next page so you can see how our modifications will change the resulting website.

Justin Beaver: Ever since I learned HTML, my life has made a complete 180

Posted by: Damian Wielgosik

Justin Beaver confessed something that even his greatest fans would have never expected of the skilled musicians and lyricist. The young rock-and-roller admitted that since he typed his first title tag, his life became easier. It has been reported by those surrounding the Canadian that Beaver's private mentors, Ryan Loseling and Nicolas Crate, often walk around Los Angeles disputing what a great tool the HTML validator is.

Justin Beaver's happy cat

Beaver has already created some websites and does not intend to stop there. "I will probably have a song about HTML on my next album," - the artist added.

As you can see, much has changed. We've added simple colors, backgrounds, changed the font style and so on. Let's proceed step-by-step for how to accomplish the effects in the image above.

The first step is to save the entire HTML code into a separate file. For me it's called article.html. Then create a separate file in which we keep our CSS rules. Let it be main.css.

The files look like this on my computer:

We can now try to open article.html in a browser and the main.css file using a text editor. I recommend Sublime Text Editor or TextMate. After each change made in main.css, we can refresh the browser the page in order to update its appearance.

We now need to open the article.html in a browser, and load code from the main.css file. This is done through the tag <link> in the <head> in the HTML code. Just add a tag like this in the head:

```
<link rel="stylesheet" href="main.css">
```

The "href" attribute indicates where the file is located. "Stylesheet" tells us that it is a CSS style sheet.

Okay, to start making changes to visual appearance in CSS, let's try to find the right selector for the heading, similar to the code for windows and walls.

```
h1 {
}
```

Here we are! It is here that we can tell the browser "for all elements in <h1>, apply the following appearances." Note that the braces are currently empty. Let's try to tell it that we want the heading text to be green. We'll apply the property "color" and set it to "green."

```
h1 {
    color: green;
}
```

The operation for this rule is explained in the diagram below:
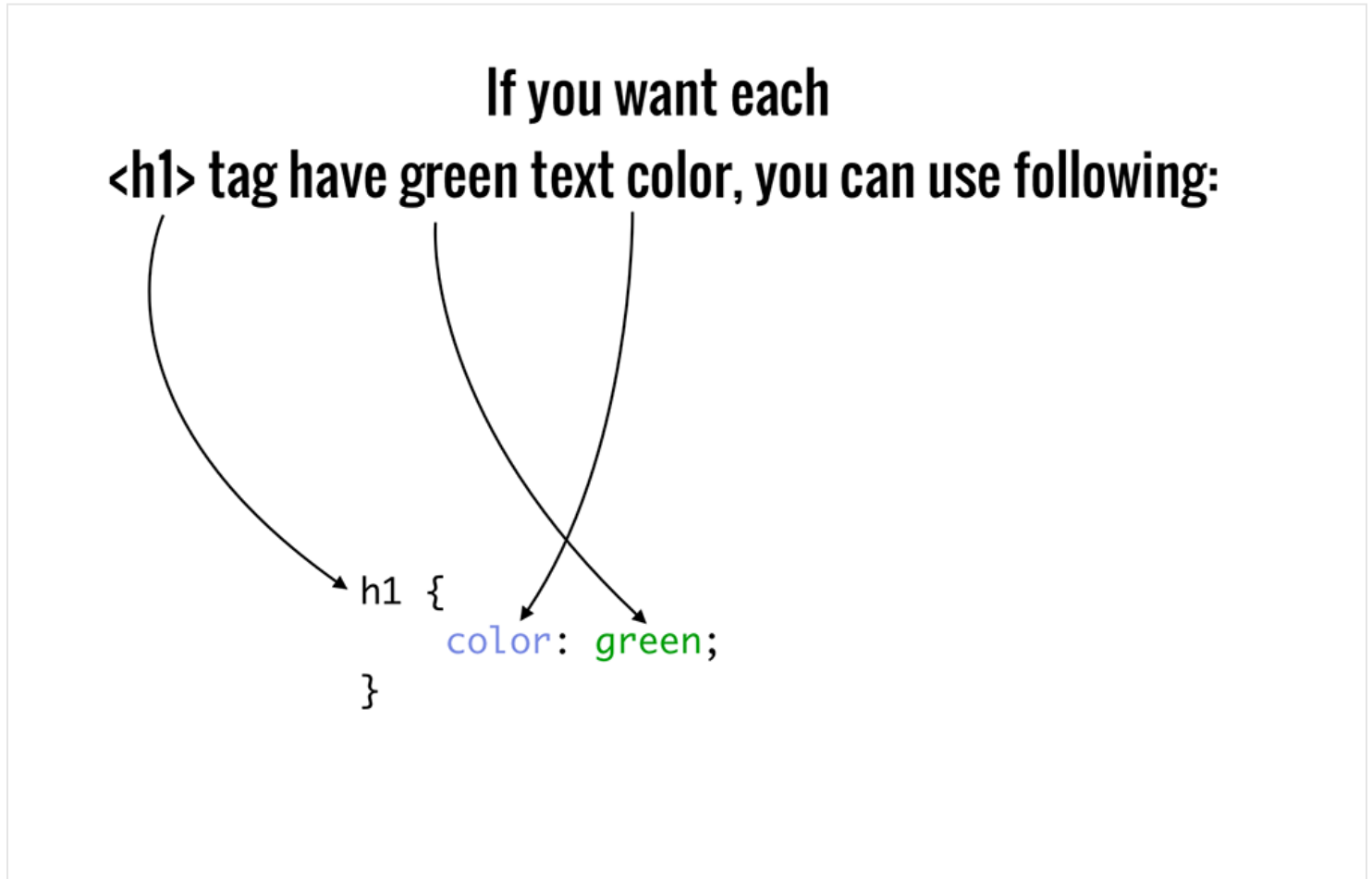
Let's check out how our page looks after the changes!

Justin Beaver: Ever since I learned HTML, my life has made a complete 180

Posted by: Damian Wielgosik

Justin Beaver confessed something that even his greatest fans would have never expected of the skilled musicians and lyricist admitted that since he typed his first title tag, his life became easier. It has been reported by those surrounding the Canadian th Ryan Loseling and Nicolas Crate, often walk around Los Angeles disputing what a great tool the HTML validator is.

Yes! The title is indicated in green.

Now we want to address the next section with information about the author. Let's say we want the text in white with a red background as shown a few pages ago. This is the current HTML code:

```
<article>
  <header>
    <h1>Justin Beaver: Ever since I learned HTML, my life has
made a complete 180.</h1>
    <p>Posted by: Damian Wielgosik</p>
  </header>
```

Let's use CSS, and find the appropriate selector ("p {") and try to give it a background red color and a white color text:

```
p {
    background-color: red;
    color: white;
}
```

Our main.css code should currently look like this:

```
h1 {
  color: green;
}

p {
    background-color: red;
    color: white;
}
```

As you can see, we add a rule one under the other. Time to see how it looks now our website...

Justin Beaver: Ever since I learned HTML, my life has made a complete 180

Posted by: Damian Wielgosik

Justin Beaver confessed something that even his greatest fans would have never expected of the skilled musicians and lyricist. The young rock-admitted that since he typed his first title tag, his life became easier. It has been reported by those surrounding the Canadian that Beaver's priva Ryan Loseling and Nicolas Crate, often walk around Los Angeles disputing what a great tool the HTML validator is.

Oops, that's not quite right. It seems all the other paragraphs have also been changed to have the new background and text color. It's a problem with our code, because we used the following:

```
p {
    background-color: red;
    color: white;
}
```

What we actually told the browser is to "find **all** <p> elements and apply changes." However, we only wanted to change the paragraph in the header line of the article.

We now need to modify the code so that the above selector to only apply to the <p> in <header>, which is a "child" of the <article>. The code should reflect this hierarchy:

```
article header p {
    background-color: red;
    color: white;
}
```

Let's see the effect of these changes.

Justin Beaver: Ever since I learned HTML, my life has made a complete 180

Posted by: Damian Wielgosik

Justin Beaver confessed something that even his greatest fans would have never expected of the skilled musicians and lyricist. The youn admitted that since he typed his first title tag, his life became easier. It has been reported by those surrounding the Canadian that Beaver'

Much better! It seems we were able to target the correct paragraph. But how did this happen? Well, we used the above code to tell the browser to know which tags the CSS selector should target. We do this by examining the HTML code and finding all the tags which should match the selector. In our case, we had nested tags of <article>, <header>, and <p>, so the CSS selector "article header p" let's us specify exactly where the changes will be applied.

Let's move on to the image in the article.

The dimensions of this article, let's say, should be 600 pixels wide. And remember that our corresponding HTML tag for the image is <figure>. Let's specify our CSS code to reflect this:

```
article figure  {
    width: 600px;
}
```

With this code, every <figure> in the <article> tag will have a width of 600 pixels. Note that the "article" distinction would be helpful if we had multiple images throughout the blog post and wanted to specify different criteria for each. But since we only have one image, let's move on to the border code:

```
article figure  {
    width: 600px;
    border: 3px solid black;
}
```

Here we've added a property called "border." After the colon, we specify the width of the border (3 pixels), and the style of the border "solid" with the color "black."

Let's see how it looks:

Justin Beaver's happy cat

It looks like we have a problem. While the border is displayed with the correct style and color, the image displays beyond our 600 pixels. This is because we established the width of the element <figure>, but the <img> tag does not have any fixed width and thus keeps its original size. It would be nice if the image took 100% of the width of its parent <figure>. This is coded very simply:

```
article figure img {
    width: 100%;
}
```

It now looks like this:

Justin Beaver's happy cat

It would be nice to add some "padding" or space between the border and image. We do this by adding the property "padding." We can modify the code as follows:

```
article figure  {
    width: 600px;
    border: 3px solid black;
    padding: 5px;
}
```

The result:
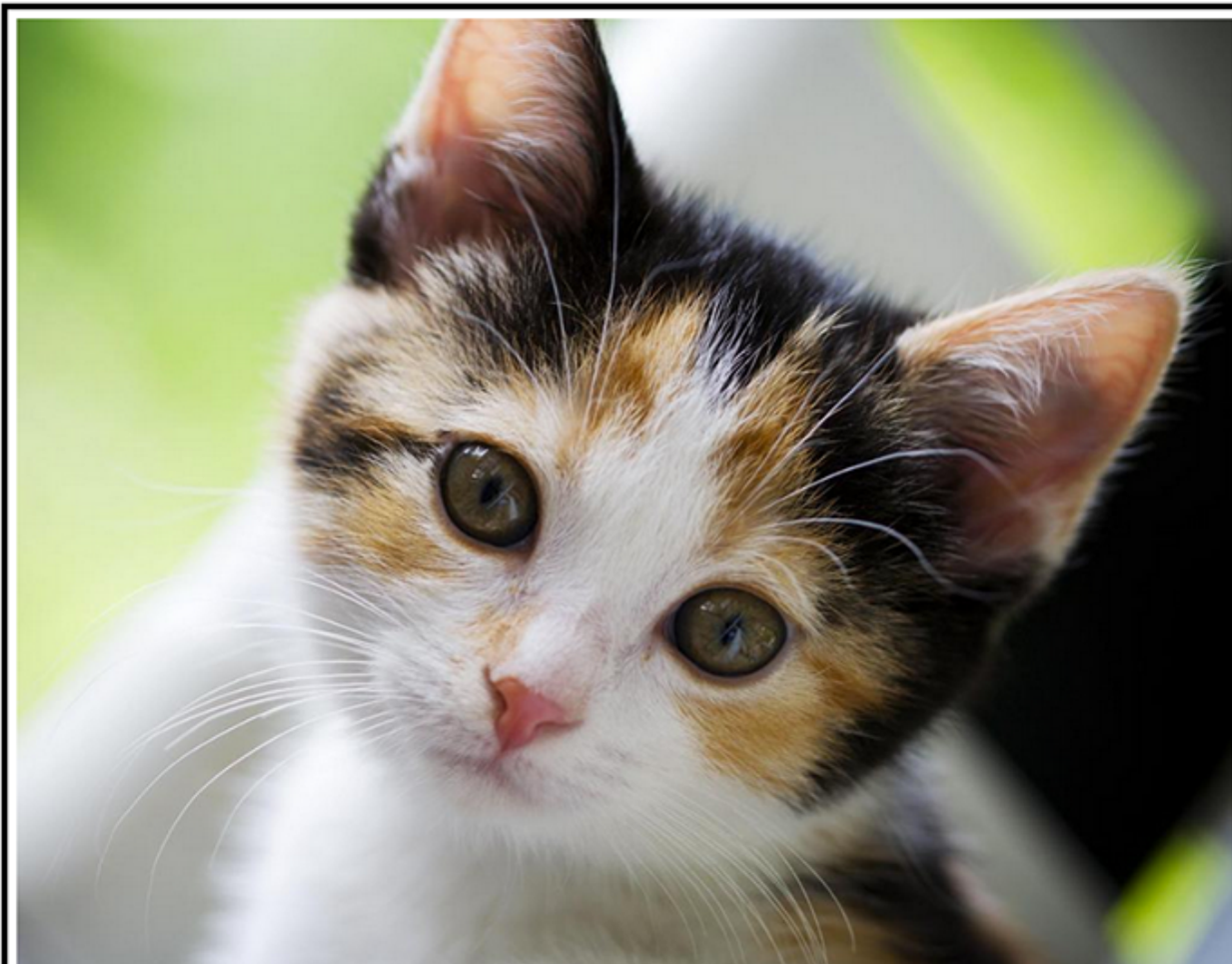
Justin Beaver's happy cat

You can try yourself to modify the value of the "padding" and see how the white gap changes between the picture and the border.

Our page is looking good now, but we're not yet finished. The current paragraph text extends almost the entire width of the browser window which isn't very readable. Perhaps it would be fitting to somehow reduce the width of the text? Maybe limit it to 800 pixels?

Let's choose a special CSS selector for this:

```
article {
    width: 800px;
}
```

That's better.

Now what about font? If you look at the original image of our site, we have a slightly different font. Just as you can edit font styles in Microsoft Word, you can edit them in CSS too. In order to specify font, you want to add this property to the highest tag so that it applies to all text within that tag. For example, we'll set the font as a property for <body>, so that every element below <body> will have this setting. In the picture, I used a font called Verdana.

Let's try to apply it:

```
body {
    font-family: Verdana;
}
```

You can see the differences by deleting this line or changing the font-family to a different style. For the header, paragraphs, etc. the browser will display everything in Verdana.

Finally, our code in the main.css file should look like this:

```css
body {
    font-family: Verdana;
}

article {
    width: 800px;
}

article header h1 {
    color: green;
}

article header p {
    background-color: red;
    color: white;
}

article figure  {
    width: 600px;
    border: 3px solid black;
    padding: 5px;
}

article figure img {
    width: 100%;
}
```

In general, it's good practice to start your code with the most general selectors and move into more complex ones. I started from body, followed by article, and so on, going from top to bottom. The higher the detail, the lower it sits in the list.

Menu, Please!

Another popular part of websites is menu. Basically, it's a list of items which are often just simple links pointing to other places on the site. Let's implement it! We will start with the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
      <meta charset="utf-8">
      <title>Menu</title>
      <link rel="stylesheet" href="main.css" media="screen">
  </head>
  <body>
    <nav>
      <ul>
        <li>
          <a href="index.html">Home</a>
        </li>
        <li>
          <a href="training.html">Training</a>
        </li>
        <li>
          <a href="conferences.html">Conferences</a>
        </li>
        <li>
          <a href="about.html">About us</a>
        </li>
      </ul>
    </nav>
  </body>
</html>
```

Our menu will consist of four items:

- Home
- Training
- Conferences
- About us

We want it to look something like this:

You might notice that under the tag, we've added the new tags <nav>, <ul>, and <li>.

<nav> is used for specifying all kinds of navigation functions on websites that contain links to internal or external information. So putting <nav> into the code says "everything inside <nav> will be used to navigate around the website."

Within <nav>, we place the <ul> tag followed by several <li> tags. The tag <ul> represents an "unordered list" (like a bullet list) and the <li> tags represent each individual component of that list (single bullet). When creating websites, it's common that an unordered list will be the most reasonable choice when it comes to mapping

menu pages. In fact, the menu is kind of a list of links that has been created without a predetermined rule as to the order of its elements.

With just the code above that is still unfinished, our list should be displayed as follows:

- Home
- Training
- Conferences
- About us

You might have seen something similar, even when creating a text document on a word processor, when you want to create a list with bullets. Without CSS `<ul>`, however, our list would simply begin with a ".". In contrast, our menu can be much more complex. We can give it a border, color, background, etc. Each link is by default displayed in blue as seen in the image above.

Let's now try to produce a more stylized menu in our CSS code.

Normally, we start from the most general tag in the HTML code, right? In this case, that top-most code begins with `<nav>` since this is responsible for our navigation menu. Within this tag, there is not much to do since this tag does not deal directly with the changes of appearances for the bullet list.

The next tag is then <ul>, which begins the unordered list. We want our list to be displayed slightly differently than the default. The most important thing is to have a new background:

```
nav ul {
   background-color: PaleVioletRed;
}
```

For the background color we chose the name PaleVioletRed. Reloading the page shows our changes as a result of adding this code.

Indeed, we've applied the color cyan as the background for the entire element of <ul>. This is because we applies it to <nav> and <ul> tags, as per the following selector.

```
nav ul {}
```

Now we want to get rid of the black, round dots in this list to make it look more like a menu. We can hide it, thanks to the property "list-style" as shown below:

```
nav ul {
    background-color: PaleVioletRed;
    list-style: none;
}
```

Setting list-style to none makes the list have no distinguishing marks.

It looks much better:
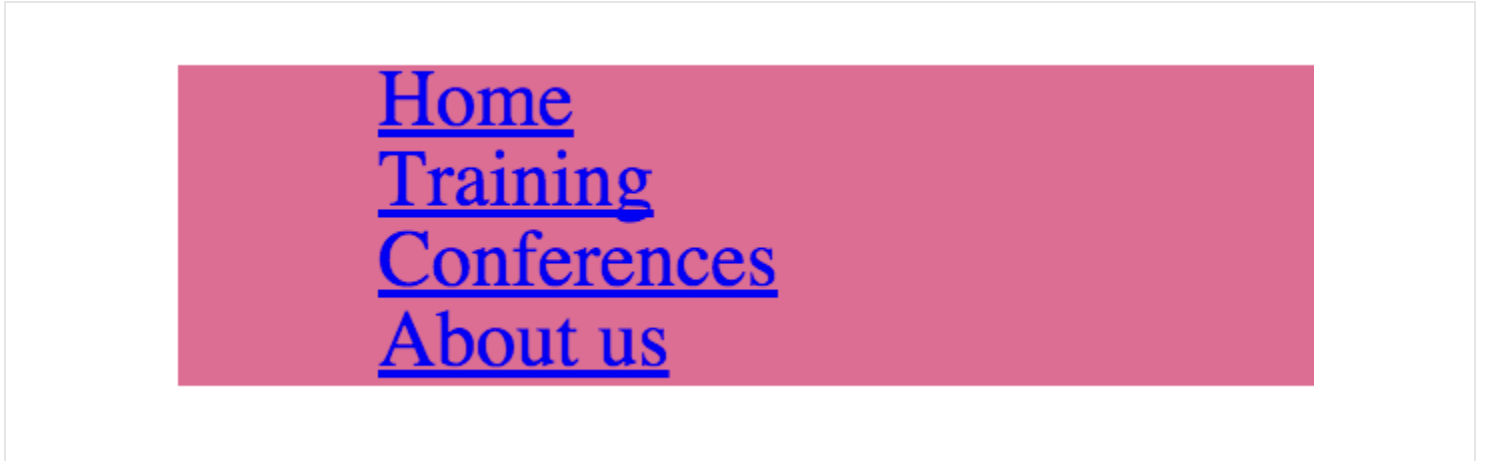
The large swath of color is surprisingly big. We're going to trim it down a bit using the same exercise as the image border earlier (e.g. padding).

```
nav ul {
    background-color: PaleVioletRed;
    list-style: none;
    padding: 0;
}
```

As you can see below, it looks much better now, slowly approaching a nice form:
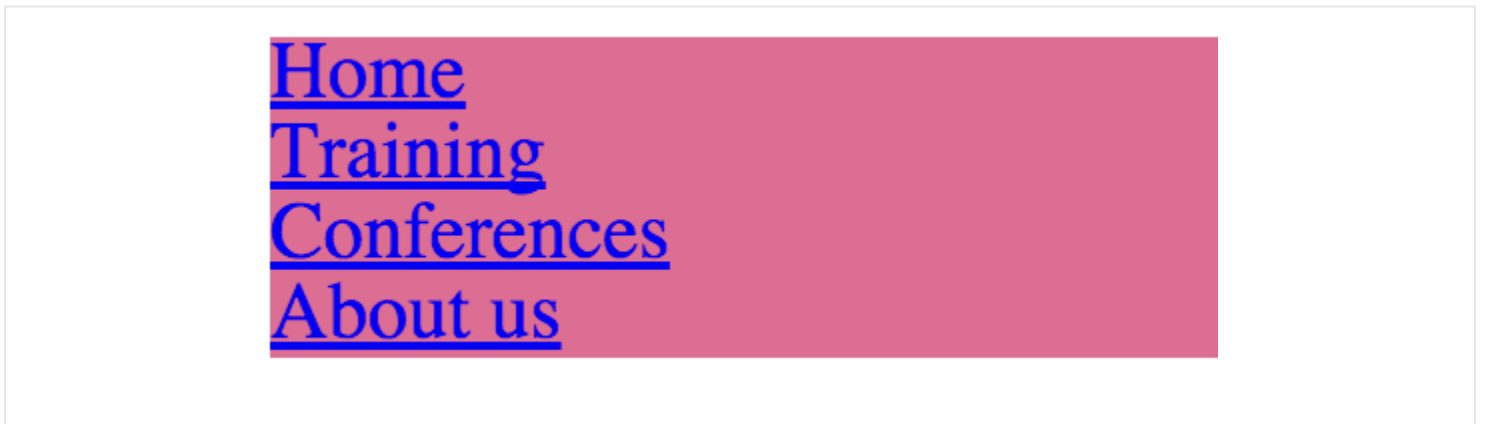
Now it's time to work on the dimensions. Our navigation has to be 200 pixels wide:

```
nav ul {
  background-color: PaleVioletRed;
  list-style: none;
  padding: 0;
  width: 200px;
}
```

At the end, we'll add a border to the list exactly like the image. It will be expressed as a solid line, with a 1 pixel width, and a "light blue" color:

```
nav ul {
  background-color: PaleVioletRed;
  list-style: none;
  padding: 0;
  width: 200px;
  border: 1px solid MediumVioletRed;
}
```

Here's the result and it's looking great!

Home
Training
Conferences
About us

So there is our beautiful outer frame. Time to frame each individual item in the list, which can be addressed using the following CSS selector:

```
nav ul li {}
```

And so this code looks at <nav>, <ul> and then <li>. What seems to be off is that each item in the list needs its own border:

```
nav ul li {
   border-bottom: 1px solid MediumVioletRed;
}
```

With this code, we've added a border-bottom, so every <li> item now has the same border type as the outer border, on the bottom portion of the text.

Currently, our menu should look like this:

We now have two problems. The first is the left-side spacing between the border and the elements list. Let's change it using our friend "padding":

```
nav ul li {
   border-bottom: 1px solid MediumVioletRed;
   padding: 5px;
}
```

It's much better, right? We've added a 5 pixel wide padding between the text and the borders.

Our second problem is less visible, but still exists as a double-line at the bottom of our menu. This is because our border code for the menu is adding to our border code for the last item when we added a "bottom-border." Remember that we used the code in <ul> of <nav> to specify a border:

```
nav ul {
  background-color: PaleVioletRed;
  list-style: none;
  padding: 0;
  width: 200px;
  border: 1px solid MediumVioletRed;
}
```

And also remember that we set the list-style to "none" so that bullets or any others signs don't appear:

```
list-style: none;
```

So by setting "none" as a value we will sort of disable a property so it won't have any graphical effect.

Let's do the same thing, only using the property "bottom-border" and setting the value to "none." However, we want to target only the last item in the menu, so that its bottom border does not conflict with the larger bottom border.

```
nav ul li:last-child {
   border-bottom: none;
}
```

The result of applying this code is super effective:
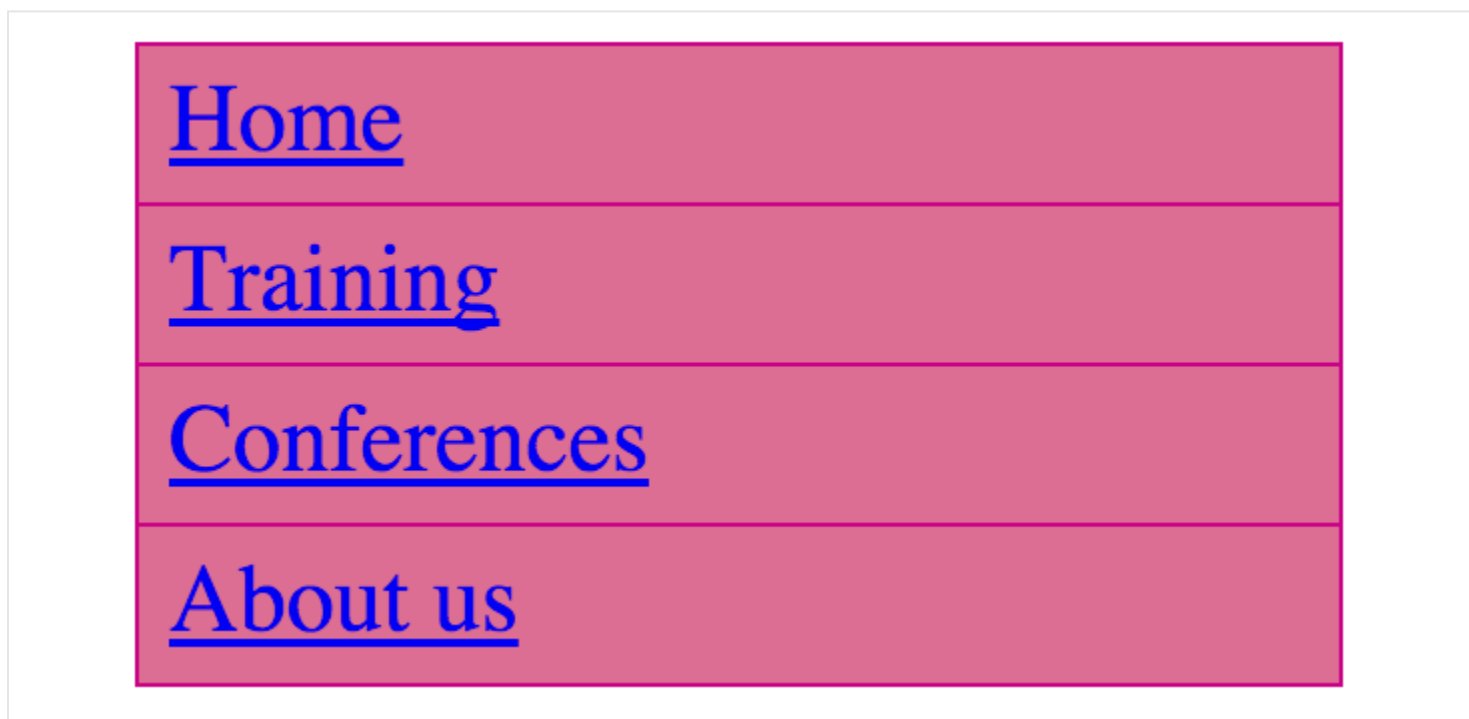
The double-border has disappeared, all because we looked at the "last-child" property in <ul> of <nav>, and then we've chosen the last <li> in it and turned off the bottom border. The pseudo-selector is "last-child" which indicates the last item in the list.

```
nav ul li:last-child {}
```

This selector can be translated as follows:

"Look at <nav>, then <ul>, and apply all changes to the "last" element.

The last thing we need to do is to adjust the text in the links. You create links in HTML as follows:

```
<a href="url">Text entered here will take you to the specified
web address</a>
```

We're using the <a> tag along with the attribute href. The value for this attribute should be the address to which you want to move the user if he or she clicks on the link. In our example, we have four links. One of them looks like this:

```
<a href="training.html">Training</a>
```

This means that the browser will show the word "Training" which can be clicked on and then sends the browser to the page that has been saved in the file "training.html."

Knowing that this tag is a part of HTML code, we can create a special CSS selector that looks specifically for this tag:

```
nav ul li a {}
```

Voilà!

Let's add new properties to our new selector. First of all, let's change the font color to white.

```
nav ul li a {
   color: white;
}
```

Refreshing the browser shows our new changes:
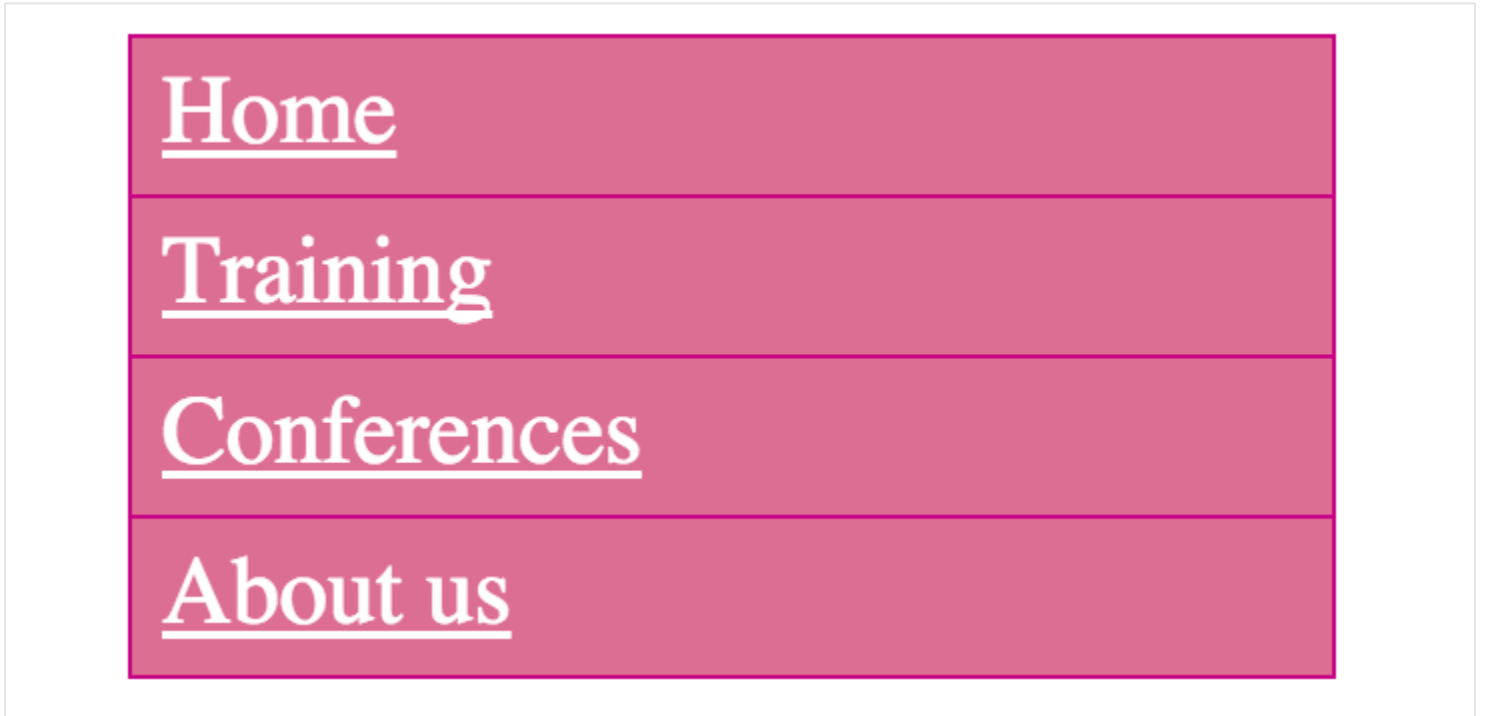
Great! We now have white color links. Now let's change some emphasis marks. The browser is set to highlight all links in the form of "text-decoration: underline" in CSS. We want to change this value, just like we did before by using the value "none."

```
nav ul li a {
   color: white;
   text-decoration: none;
}
```

Beautiful! We have completed the menu that we wanted.

As a side note, if you are working with many links, you might remember that on many pages when you hover over a link, the text becomes emphasized somehow.

Check out this link I posted on my Twitter (without an underline):

When a mouse hovers over this link, something interesting happens that many internet users know well—the text becomes emphasized, or in this case, underlined:

Damian Wielgosik @varjs · Jan 31

Let's imagine what would happen, if JavaScript was a human interviewed by a random news reporter.
ferrante.pl/frontend/javas... #javascript

Let's try to do something similar in our menu that will allow a link to be highlighted or emphasized whenever we hover over it. We'll use a pseudo-selector called "hover":

```
nav ul li a:hover {
    text-decoration: underline;
}
```

This time we added it to the links <a>. This means when you mouse over a link, the effect will be applied. This applies to hovering over other elements as well:

```
div:hover
li:hover
img:hover
```

The effect is seen below when we hover over the "Conferences" link.
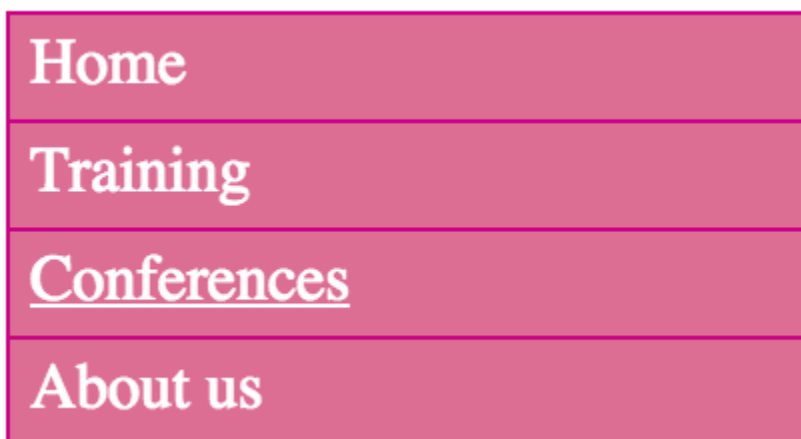
In summary, the final CSS code should look like this:

```css
nav ul {
  background-color: PaleVioletRed;
  list-style: none;
  padding: 0;
  width: 200px;
  border: 1px solid MediumVioletRed;
}

nav ul li {
  border-bottom: 1px solid MediumVioletRed;
  padding: 5px;
}

nav ul li:last-child {
  border-bottom: 0;
}

nav ul li a {
  color: white;
  text-decoration: none;
}

nav ul li a:hover {
  text-decoration: underline;
}
```

The newly emerged pseudo-selectors (last-child and hover) will be useful in the future.

By the way, in this chapter you have learned how to use links and put them into HTML documents. At this point, we have used only addresses pointing to the local files (like training.html) saved on your computer, however you can also use links referring to external websites living on the web like this:

```html
<a href="http://twitter.com/varjs">My Twitter</a>
```

The above code in the browser will be displayed as My Twitter. Note that the address contains "http://" at the very begininng. It's a rule to learn that every link you used in a HTML document that points to another website, must be prefixed by "http://". Otherwise, your links won't redirect users to the places they should.

Understanding Selectors in CSS

The example of the menu, and the two pseudo-selectors will allow us to learn one more important concept. You may have noticed that each selector our CSS code consists of the same names as their respective HTML tags. So, whenever we change something in the structure of code we must also change the CSS. For example, sometimes it might be necessary to no longer have a heading:

```
<h1>Justin Beaver shaved his mustache</h1>
```

Let's instead use the element <p>:

```
<p>Justin Beaver shaved his mustache</p>
```

With this change, we run the risk of errors in the display. To minimize this risk, we use CSS classes and IDs.

Every HTML element may have a special attribute named "class," where we can provide one or more class names. Here are a few examples of how you can add this attribute:

- <p class="news-item"> – A class called "news-item"
- <li class="menu-item"> – A class called "menu-item"
- <article class="news"> – A class called "news"
- <q class="important-quote"> – A class called "important-quote"

And so on. Note that class names are arbitrary, and must be single words or "clusters" using dashes like in the example of "my-item" or underscores like "special_item".

You can also add multiple class names for a single tag:

- <p class="news-item special-info"> – Class names "news-item" and "special-info"
- <li class="menu-item selected"> – Class names "menu-item" and "selected"

You might be wondering what this means and why classes are useful. Suppose that we create a page with information on sports news. Let us assume that we have a list of news and we want to highlight one item (I've highlighted one line of our list below). The code would look something like this:

```
<article><p>normal news</p></article>
<article><p>special news</p></article>
<article><p>normal news</p></article>
<article><p>normal news</p></article>
<article><p>normal news</p></article>
```

If we wanted to write CSS code that distinguishes a special background for only the features news, we have a problem. Our expertise so far allows us to "style" all the elements in <article> in the form of the selector:

```
article {}
```

But this will look for all <article> elements. CSS classes allow us to avoid this confusion. Let's describe each of the <article> elements with a class that corresponds to its importance for the entire site. So, let's call the important news "main-news" and the rest "normal-news" as shown in the HTML code below:

```
<article class="normal-news"><p>normal news</p></article>
<article class="main-news"><p>special news</p></article>
<article class="normal-news"><p>normal news</p></article>
<article class="normal-news"><p>normal news</p></article>
<article class="normal-news"><p>normal news</p></article>
```

Now, using CSS, we can use the following selectors:

```
.main-news {
  background-color: LightBlue;
}

.normal-news {
  background-color: LightYellow;
}
```

For each class, note that they begin with a dot "." followed by the class name. So we can use this construction to target selector elements of the same class in HTML. The "main news" will now have a background color of light blue, while a normal news is coloured in LightYellow.

In the browser, this code looks something like this:

normal news

special news

normal news

normal news

normal news

Thanks to CSS classes, we can denote characteristics for specific elements on the page, that allows us to simultaneously target and ignore elements with the same tags. If we ever wanted to change `<article>` tag to something else, let's say `<p>`, we wouldn't need to modify our CSS code as well. That's a huge benefit.

The essence of classes is explained in the following diagram:

Whenever you define a CSS code for .content like following

```
.content {
    width: 300px;
}
```

It means that a browser will find all items with class="content":

```
<div class="content">something</div>
```

and display it with width equal 300px.

In addition to classes, we also have identifiers. We use the attribute called "id" and give it a value, very similar to how classes work. An example of an ID in HTML might look like this:

```
<p id="main-content"></p>
```

The CSS code for identifiers looks something like this:

```
#main-content {
    background-color: red;
}
```

Instead of a "." before the element, we use a hashtag "#" for identifiers, followed by the element name. The code above specifies that elements with the ID "main-content" should have a red background.

It's very important to remember that identifiers are unique, so the identifier can only be used once in an HTML document. The use of the attribute "id" should be preceded by careful planning and analysis. It's also good practice to not abuse "id" usage, as it is rare that good sites consists of several unique elements.

The operation of identifiers is summarized in the following diagram:

With our new knowledge of classes and identifiers, let's rewrite our menu code to make it more robust against changes in the HTML file. Perhaps in the future we decide that we would like to make it using other tags than <ul> and <li>. If we use the appropriate classes, then we can sleep peacefully without having to worry about changing the CSS code in parallel.

Let's start by modifying our HTML. Currently, our code for the navigation menu currently looks like this:

```
<nav>
  <ul>
    <li>
      <a href="index.html">Home</a>
    </li>
    <li>
      <a href="training.html">Training</a>
    </li>
    <li>
      <a href="conferences.html">Conferences</a>
    </li>
    <li>
      <a href="about.html">About us</a>
    </li>
  </ul>
</nav>
```

Let's introduce a class. We'll give the entire container of the menu (tag <ul>) a class, and each item under it (tag <li>) a "child" class.

```
<nav>
  <ul class="site-nav">
    <li class="site-nav-item">
      <a href="index.html">Home</a>
    </li>
    <li class="site-nav-item">
      <a href="training.html">Training</a>
    </li>
    <li class="site-nav-item">
      <a href="conferences.html">Conferences</a>
    </li>
    <li class="site-nav-item">
      <a href="about.html">About us</a>
    </li>
  </ul>
</nav>
```

For this menu, I've given the unordered list <ul> the "site-nav" class, while each <li> item has the class "site-nav-item."

Time to customize the CSS code:

```css
.site-nav {
  background-color: PaleVioletRed;
  list-style: none;
  padding: 0;
  width: 200px;
  border: 1px solid MediumVioletRed;
}

.site-nav .site-nav-item {
  border-bottom: 1px solid MediumVioletRed;
  padding: 5px;
}

.site-nav .site-nav-item:last-child {
  border-bottom: 0;
}

.site-nav .site-nav-item a {
  color: white;
  text-decoration: none;
}

.site-nav .site-nav-item a:hover {
  text-decoration: underline;
}
```

To better understand what has changed, look at the comparison of the old and new CSS code:

```
1   nav ul {
2     background-color: PaleVioletRed;
3     list-style: none;
4     padding: 0;
5     width: 200px;
6     border: 1px solid MediumVioletRed;
7   }
8
9   nav ul li {
10    border-bottom: 1px solid MediumVioletRed;
11    padding: 5px;
12  }
13
14  nav ul li:last-child {
15    border-bottom: 0;
16  }
17
18  nav ul li a {
19    color: white;
20    text-decoration: none;
21  }
22
23  nav ul li a:hover {
24    text-decoration: underline;
25  }
```

```
1   .site-nav {
2     background-color: PaleVioletRed;
3     list-style: none;
4     padding: 0;
5     width: 200px;
6     border: 1px solid MediumVioletRed;
7   }
8
9   .site-nav .site-nav-item {
10    border-bottom: 1px solid MediumVioletRed;
11    padding: 5px;
12  }
13
14  .site-nav .site-nav-item:last-child {
15    border-bottom: 0;
16  }
17
18  .site-nav .site-nav-item a {
19    color: white;
20    text-decoration: none;
21  }
22
23  .site-nav .site-nav-item a:hover {
24    text-decoration: underline;
25  }
```

Try to compare each selector. As you can see, we traded the tag names for class names, which gives us much more flexibility in writing code. Note that they are shortened by removing the redundant "nav" tag.

It's important to try to use class names as selectors instead of tags, unless it is obvious like <a> for addresses. This cannot be replaced with any other HTML tag. Identifiers are only used when the item is unique in the page. In this cause, it could be just the menu, but oftentimes navigation appears several times on a website (for example at the bottom and top).

The operation and definition of classes and IDs are summarized in the following diagram:

you can have multiple elements with the same .className   ⟶

```
<div class="news">news</div>
<div class="news">other news</div>
<div class="news">another news</div>
```

you apply styles to elements with class names
using dot together with a class name   ⟶

```
.news {
    width: 300px;
}
```

an id attribute is unique per document
and cannot be used twice   ⟶

```
<article id="article">article</div>
```

you apply styles to elements with an id
using # sign together with an id   ⟶

```
#article {
    width: 300px;
}
```

Summary Time!

Time to rest a moment and reflect on what we've learned so far. By now, you've met many of the basic concepts required to create web pages.

We began by understanding how websites are built, with plain text that describes how individual parts work and compared them to how blocks are stacked one on top of the other.

With this analogy, we found that blocks correspond to HTML tags. All HTML tags are predefined and definitions, best uses, and synergies between other elements are standardized by the W3C organization. For example, if we need a paragraph of text, we use the <p> block. If we need an article, we use the <article> block. If we're dealing with headers, we use hierarchies of <h1>, <h2>, <h3>, <h4>, <h5> or <h6> depending on the context. We know that the entire list of HTML tags can be found on the HTML specification.

We found that tags can be nested within one another. For example, the article for our page that consisted of a title, image, and paragraphs. These elements were denoted by <header>, <h1>, <figure> and <p>, which were then nested under the "parent" tag <article>. In HTML we try to think hierarchically. We build large blocks with smaller blocks and always put the most important blocks higher in the code than the less important blocks.

Then we found that websites with only this HTML code are visually unattractive. With CSS, we learned that we can target HTML tags and decorate them with special

selectors. We know that if we wanted to get "all the paragraphs in an <article>, our CSS selector should be written as with properties and values that we want within the brackets.

```
article p {}
```

In addition, we learned that classes are a helpful component of CSS. Thanks to classes, we can modify HTML tags to be more robust so that we don't always have to go to the CSS file and make changes when the HTML changes. Each tag in HTML can be given a class attribute, and these class names can be whatever you choose, but they should describe the function that it plays in the document. For example, in a press release, we can write this in HTML:

```
<article class="news">...</article>
```

The CSS code then should reflect those classes:

```
.news {}
```

In this way, we have learned the most important fundamentals of HTML and CSS. When building web pages, we always divide content into smaller parts and determine the function of elements. We try to be as specific as possible for even the smallest element, being able to code and describe things like menus, articles, titles, dates, paragraphs, quotes and pictures.
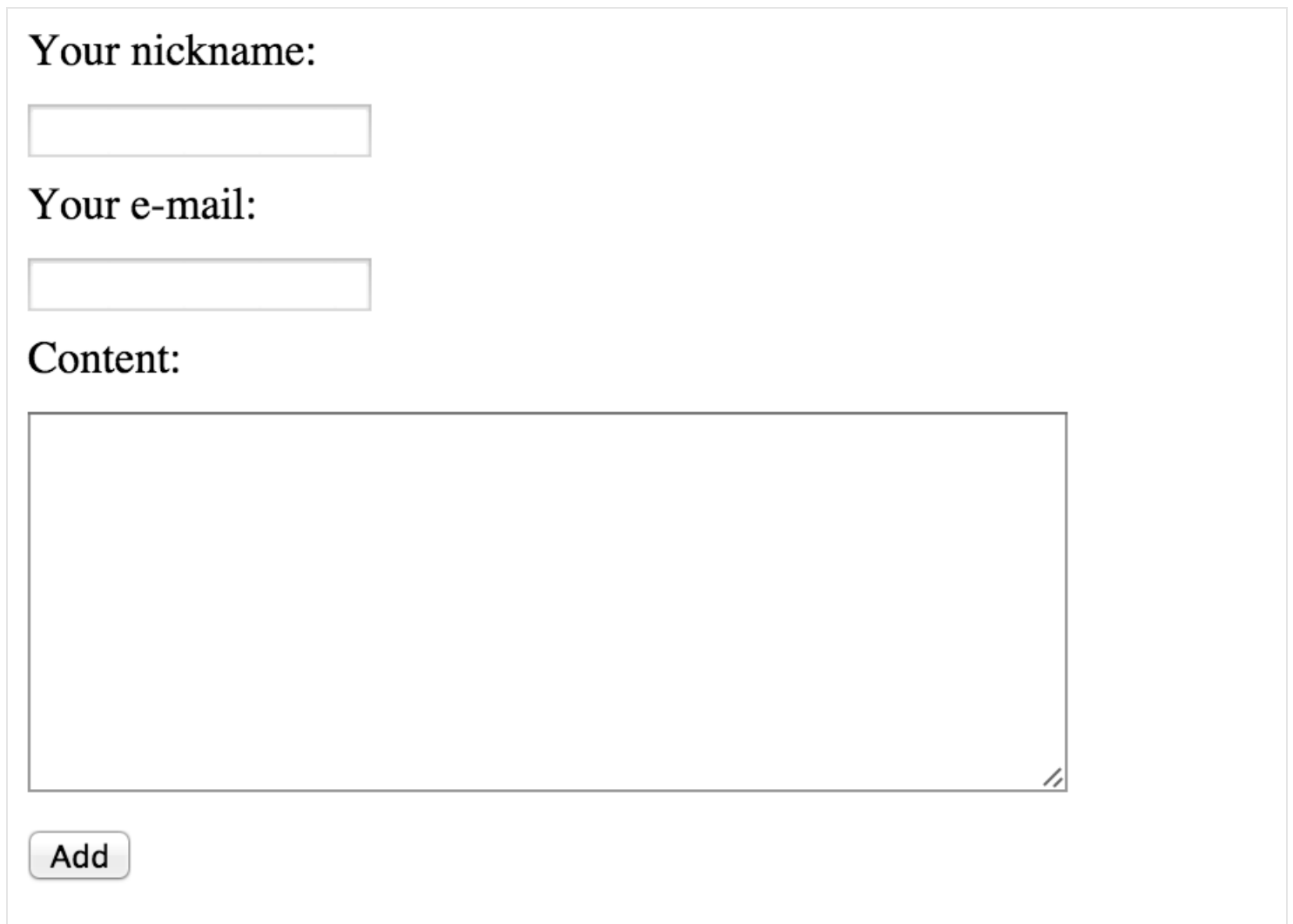
Always remember that HTML is a data layer while CSS is a presentation layer. It means, that HTML stores data information described by the use of proper tags, when CSS code has to visualize this data adding colors, fonts, borders and so on.

At this point you have probably asked yourself how one can publish a page on the web so your friends can see it. Being more precise, how does it happen that after typing functionite.com in a browser address bar, you actually see a website? It's simple. First off, you need to buy a domain which will be a virtual address of your own website. In above example, I had to register functionite.com. There are a lot of companies that make this process easy. All you need to know is a unique domain name that will be pointing to your website. After you have a unique id where your website is going to be available at, you need a web server to where you will upload your complete website. All images, CSS, HTML and other files must be first uploaded to that server, which is actually a special computer connected to the internet, that makes your website be accessible from any place in the world. Again, the web servers business is a huge market and you will certainly find a decent company that you can purchase a server space from. There are also companies that offer easy deals so you can purchase a domain and a web server access at once.

Forms in HTML5

So far, we've managed to create a few interesting versions of the website. We haven't yet covered the forms for which users can enter data. Forms are used across the Internet everywhere; search engines like Google or Bing, Facebook status fields, or Gmail email editors allow you to type or send information.

Let's make a simple form to allow users comment on our article. We want the form to look something like the following image:

Let's now select the individual components that make up the form. We will use the same selection colors for elements that perform a similar function:

As you can see, blue indicates names or description for each of the fields. Green elements show places where you can enter one-liner text. The orange area lets you enter longer chunks of text, and purple is used for a button to send the form. For each group, we'll use the same tag.

For names we'll use <label>. For shorter fields – <input type="text">. The longer texts will use <textarea>. Buttons are made with <input type="submit">. These are the most popular elements of HTML, that are used to build forms on websites. As always, we start with a blank HTML template page and add then add more elements.

```
<!DOCTYPE html>
<html lang="en">
   <head>
       <meta charset="utf-8">
       <title>Form</title>
       <link rel="stylesheet" href="main.css">
   </head>
   <body>
   </body>
</html>
```

We need to use the appropriate tag that tells the browser, "Hey, the form starts here!" This is very similar to the tag <article> for indicating where an article element starts). In HTML, we use the <form> tag for indicating a form element

```
<!DOCTYPE html>
<html lang="en">
   <head>
       <meta charset="utf-8">
       <title>Form</title>
       <link rel="stylesheet" href="main.css">
   </head>
   <body>
     <form>
     </form>
   </body>
</html>
```

Now, we want to add the first description name: "Your nickname." From here on, we'll examine the code fragments that are contained under <form>, because the rest of the HTML is already well known to you.

```
<form>
   <label for="nickname">Your nickname:</label>
   <input type="text" id="nickname" name="nickname">
</form>
```

Here we have two <label> elements which give the description name, and the <input> elements for text input. The <input> tag has three attributes: input type with a "text" value, name with "nickname" value and id also with "nickname" value. The "type" attribute's value means that it's a short text field.

The field below appears for these types of fields:

You'll use <input type="text"> in your code to indicate that a user can type within the field. Note that text fields can be styled using CSS. We can change its border, width, or distance between any text and a text field border. A rough example is shown below:

The "name" attribute is used to name each of the fields. This is useful when you send the form to the server. It helps to identify which value comes from which field.

Note also the correlation between the value of the "id" and the value of "for" attribute in <label>:

```
<label for="nickname">Your nickname:</label>
<input type="text" id="nickname" name="nickname">
```

In the "for" attribute, you should use the id of the field described by the <label> element.

Let's make another form, now for the "email" field.

```
<form>
  <label for="nickname">Your nickname:</label>
  <input type="text" id="nickname" name="nickname">
  <label for="user-email">Your e-mail:</label>
  <input type="email" id="user-email" name="user-email">
</form>
```

The only difference here is that the "type" attribute has an "email" value. The meaning is, of course, so that the user can enter their email. Note that anything typed within the "email" field will have to be validated as a correct e-mail address. If it's not a valid email, the browser will display an error message and will not send the form.

Another field we need to add is a place for the comment. For longer texts, we use the <textarea> tag:

```
<form>
  <label for="nickname">Your nickname:</label>
  <input type="text" id="nickname" name="nickname">
  <label for="user-email">Your e-mail:</label>
  <input type="email" id="user-email" name="user-email">
  <label for="content">Content:</label>
  <textarea rows="10" cols="50" id="content"  name="content"
></textarea>
</form>
```

Note that we used two new attributes: rows and cols. The "rows" attribute is used for the number of text lines that can be entered into the field until a scrollbar appears. "cols" is used for the number of character in each line. Try changing the values for yourself and see how the entire field expands or contracts accordingly.

The last thing we need to add is a button for sending the form.

```
<form>
  <label for="nickname">Your nickname:</label>
  <input type="text" id="nickname" name="nickname">
  <label for="user-email">Your e-mail:</label>
  <input type="email" id="user-email" name="user-email">
  <label for="content">Content:</label>
  <textarea rows="10" cols="50" id="content"  name="content"
></textarea>
  <input type="submit" value="Add">
</form>
```

The element <input> has the attribute type equal to submit. Whatever is typed into the value attribute will display as text on our website button.

Our code for the form now looks like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
      <meta charset="utf-8">
      <title>Formularz</title>
      <link rel="stylesheet" href="main.css">
  </head>
  <body>
    <form>
      <label for="nickname">Your nickname:</label>
      <input type="text" id="nickname" name="nickname">
      <label for="user-email">Your e-mail:</label>
      <input type="email" id="user-email" name="user-email">
      <label for="content">Content:</label>
      <textarea rows="10" cols="50" id="content"  name="content"
></textarea>
      <input type="submit" value="Add">
    </form>
  </body>
</html>
```

Our browser displays it as:

Your nickname: [   ]   Your e-mail: [   ]   Content: [   ] [Add]

It doesn't look nice. In the next chapter we will figure out how to fix it.

Differences Between <div> and <span>

The elements from last chapter appear one after the other on the same horizontal line. These elements are behaving as plain text, and will just be displayed one next to the other, even though they have certain sizes and other properties that don't apply to texts. It shouldn't be surprising, as <label>, <input>, and <textarea> are all elements of one group called inline-block. Inline-block elements can have different sizes, however browser will always display them horizontally just as a text. In this chapter we will learn about different groups of elements according to how they are laid out on the page.

First, let's figure out how one makes elements display one after the other vertically. We basically need to tell the browser, "Hey, we want a container which can be vertically stacked". Fortunately, this container tag is known as <div>, and will sort of "break" content to new lines (like page breaks or line breaks in word processors).

To see how the <div> elements work, let's use three of these and let's set them width, height, and background.

Under the assumptions given above, the HTML will look like this:

```
<div class="container"></div>
<div class="container"></div>
<div class="container"></div>
```

In CSS, we write the class ".container" when referring to the <div> tag.

```
.container {
  width: 200px;
  height: 200px;
  background-color: blue;
}
```

Let's see this in a browser:
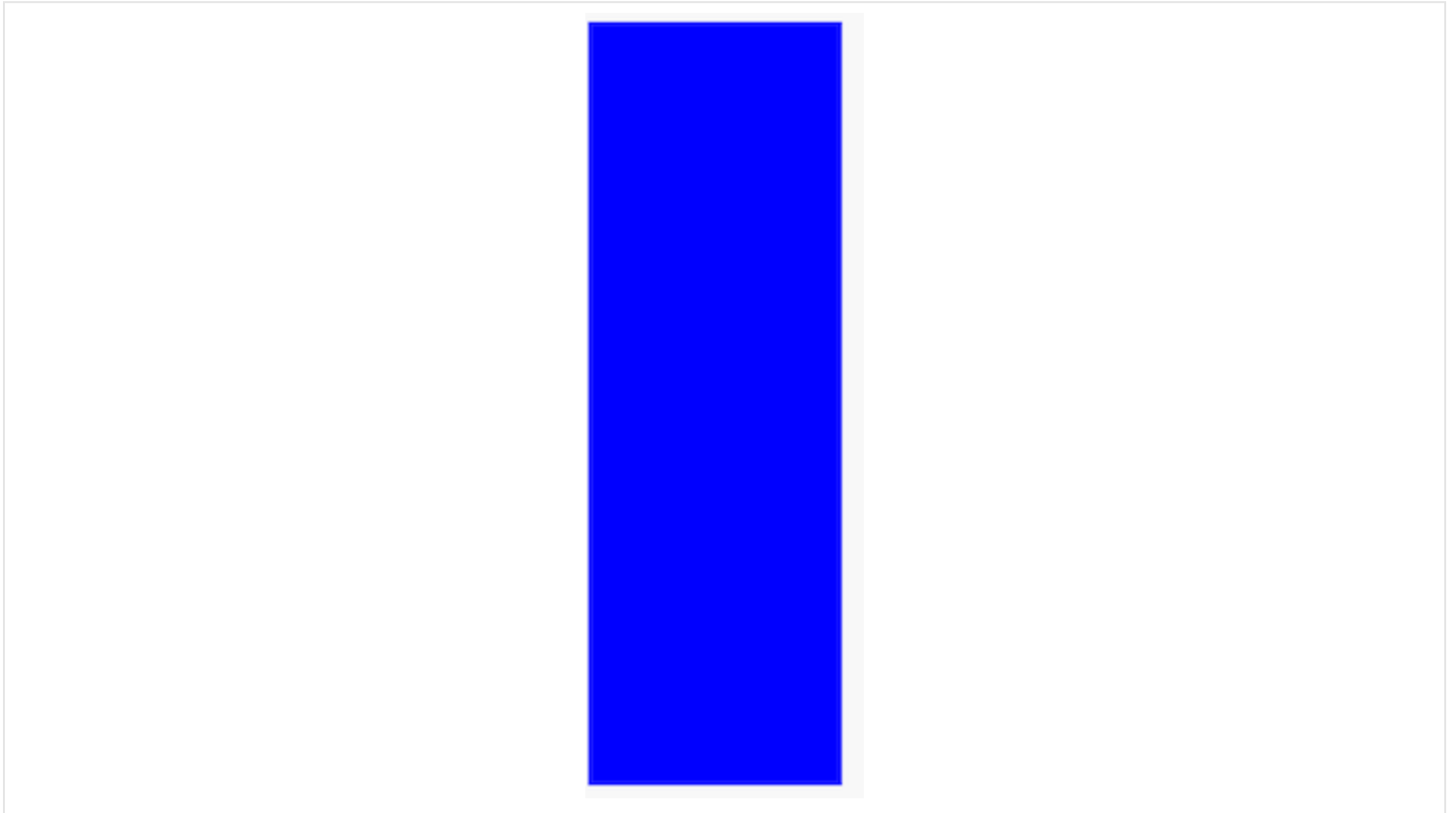
Looks like a one big rectangle. What about separating <div> tags 10 pixels from the bottom?

```
.container {
  width: 200px;
  height: 200px;
  background-color: blue;
  margin-bottom: 10px;
}
```

I added a margin-bottom property (highlighted above) to separate each <div> by 10 pixels from the bottom. Now it's time to check the effect in the browser:

Great! The browser displays these "blocks" one by one, in contrast to our previous example of a web form where the elements appeared on the same line.

Why is this different? When it comes to the display of tags, the browser recognizes three groups of elements:
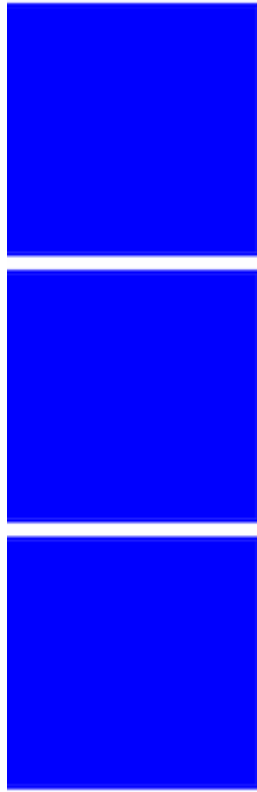
- inline
- block-level
- inline block-level

Inline elements do not cause transitions to a new line, but will be displayed one next to the other horizontally. Block elements are set like blocks that stack on top of each other and will never display next to one another horizontally, unless we use magic tricks in CSS (which we'll learn in the next chapter). Inline blocks will act as inline elements (elements are displayed next to each other), but differ in that they can be for instance resized. For example, the <textarea> field can be displayed as a large rectangle, but it can take up space beyond a single line of text.

Let me share with you some examples of items that can be classified according to one of the three groups.

- inline – <span>, <em>, <strong>
- block-level – <div>, <p>, <article>
- inline block-level – <input>, <textarea>

By default, inline elements have CSS display property set to inline. For block-level elements, its value is "block", and for inline block elements it's "inline-block". So you can explain that <span> tag doesn't cause breaking the text to the new line, because it's an inline element which means in CSS it has a property "display" set to "inline". So, considering the below code:

```
<span>one</span> <span>two</span> <span>three</span>
```

The browser will display above code in one line:

one two three

However, it's possible to change this behavior by adding one line of CSS:

```
span {
   display: block;
}
```

Now, our <span> tags display differently, each one in a new line, since we have set their display property to block:

one

two

three

For this example, we use <div>. You may wonder what this tag describes within the document. The short answer is nothing. We use the <div> tag in cases where all other tags do not find a use for what we've put in the document. It is common, therefore, that the <div> tag (in conjunction with other classes) is used to achieve different visual effects, as a result of it having no function for describing content. For example, if you want to make three vertical columns on our website, we might need an item that can separate some elements of inline or inline block elements.

In general, it's a good idea to not overuse the <div> tag if possible.

With our new found knowledge of blocks, let's make up our form code so that the form fields and their descriptions are displayed nicely one on top of the other:

```
<form>
  <div>
    <label for="nickname">Your nickname:</label>
  </div>
  <div>
    <input type="text" id="nickname" name="nickname">
  </div>
  <div>
    <label for="user-email">Your e-mail:</label>
  </div>
  <div>
    <input type="email" id="user-email" name="user-email">
  </div>
  <div>
    <label for="content">Content:</label>
  </div>
  <div>
    <textarea rows="10" cols="50" id="content"  name="content"
></textarea>
  </div>
  <div>
    <input type="submit" value="Add">
  </div>
</form>
```

It looks a bit more complicated, but the only thing we've done here is add <div> tags to surround each of the elements on the form.

Once again, let's look at a code comparison of our new code (left) against the old one (right):

```
1   <form>
2     <label for="nickname">Your nickname:</label>
3     <input type="text" id="nickname" name="nickname">
4     <label for="user-mail">Your e-mail:</label>
5     <input type="email" id="user-email" name="user-email">
6     <label for="content">Content:</label>
7     <textarea rows="10" cols="50" id="content" name="content"></textarea>
8     <input type="submit" value="Add">
9   </form>
10
```

```
1   <form>
2     <div>
3       <label for="nickname">Your nickname:</label>
4     </div>
5     <div>
6       <input type="text" id="nickname" name="nickname">
7     </div>
8     <div>
9       <label for="user-email">Your e-mail:</label>
10    </div>
11    <div>
12      <input type="email" id="user-email" name="user-email"
          required>
13    </div>
14    <div>
15      <label for="content">Content:</label>
16    </div>
17    <div>
18      <textarea rows="10" cols="50" id="content" name="content">
19      </textarea>
20    </div>
21    <div>
22      <input type="submit" value="Add">
23    </div>
24  </form>
```

And now see how the new code displays in your browser!

Your nickname:

[   ]

Your e-mail:

[   ]

Content:

[   ]

[Add]

Our form looks excellent now!

Form Extras

I have a few more surprises in store for you. We can now use a number of attributes that will help make our form more attractive. One of them is a placeholder text. Let's add it to one of the form fields, <textarea>:

```
<textarea rows="10" cols="50" id="content" name="content"
placeholder="Enter a comment here. Be nice and
polite!"></textarea>
```

The effect?

Content:

> Enter a comment here. Be nice and polite!

As you can see, in the middle of <textarea>, text has appeared with an additional description of this field, which corresponds to what we put in the attribute "placeholder". Peace of mind, when you click this text will disappear and you will be able to enter your own content. If you leave the field blank is also nothing will

happen - the contents of the "placeholder" will not be sent away, it serves only to hint to the user. This attribute can also use elements <input>.

Another very useful attribute is required, which is added to the form elements without any value. For example, an e-mail will look like this:

```
<input type="email" id="user-email" name="user-email" required>
```

Now, when we leave the field blank, when you send the form we will see the error:

There is plenty of useful attributes you can use together with form elements. Let's consider Google search input for a moment:

It's encoded per the following (as of April 2015):

```
<input id="q" aria-hidden="true" autocomplete="off" name="q"
tabindex="-1" type="url" jsaction="mousedown:ntp.fkbxclk"
style="opacity: 0;">
```

As you can see, the element input has a bit more than the usual attributes we used. Try for yourself to find the HTML specifications for each of them. Even Google will help you here :).

Attributes are also useful for transmitting elements from CSS. For example, each element input with the attribute "type" equal to "text", as shown below:

```
input[type="text"] {
  border: 1px solid blue;
}
```

In the example above, each element input has the attribute "type" equal to "text" with the value of a blue frame:

Your nickname:

Similarly, we can use the same for the email field:

```css
input[type="text"] {
  border: 1px solid blue;
}

input[type="email"] {
  border: 1px solid blue;
}
```

Everything should now look like this:

Your nickname:

Your e-mail:

Content:

Enter a comment here. Be nice and polite!

Add

As you can see, we've created two selectors in CSS which give the same value, specifically for "border." If we wanted to use something like this for a larger number of elements, we would probably copy one of those selectors and modify it slightly. As a result, we would have more independent selectors doing the same thing (i.e. setting borders). In CSS, you can shorten the operation, listing selectors after a comma to form a group.

```
input[type="text"],
input[type="email"] {
  border: 1px solid blue;
}
```

You can see that we saved space and simplified our code to have better clarity. Using our previous example, all combinations are possible selectors that we want to group:

```
.news-item,
article p,
#main header h1 {
   padding: 5px;
   margin: 5px;
}
```

Always keep in mind grouping selectors allows you to repeat certain values for several elements. This way, you have one piece of code for multiple elements in the HTML, and you won't have to look too far in the CSS file to make modifications.

Multi-Column Layout

In the previous example, we learned, among other things, the `<div>` tag. You know that they have no semantic meaning and serve primarily as a container for other blocks. Usually you add them to apply various visual changes through CSS for larger parts of the website that resemble boxes or rectangles.

So far, we have managed to get a view of the article, create a menu, and a comment form. Let's put them all together in a 3-column layout with the menu on the left, article and comment forms in the middle, and a list of materials related to the article on the right. Let's assume that the whole of this page is 960 pixels wide, centered.

Home

Training

Conferences

About

Justin Beaver: Ever since I learned HTML, my life has made a complete 180

Posted by: Damian Wielgosik

Justin Beaver confessed something that even his greatest fans would have never expected of the skilled musicians and lyricist. The young rock-and-roller admitted that since he typed his first title tag, his life became easier. It has been reported by those surrounding the Canadian that Beaver's private mentors, Ryan Loseling and Nicolas Crate, often walk around Los Angeles disputing what a great tool the HTML validator is.

Justin Beaver's happy cat

Beaver has already created some websites and does not intend to stop there. „I will probably have a song about HTML on my next album," - the artist added.

Leave a comment

Your nickname:

Hi! My name is John Doe and I would like to invite you to join me in my amazing journey thorugh life.

Let's try again to identify what we see in the graphic, functional parts, which are mapped by HTML. We have the left column which contains the menu. The central column contains the main content of the page. The right column contains a short text.

Let's start from the container that holds these three columns. As we noted earlier, we want all of them to be contained within a space no wider than 960 pixels. HTML does not have social tags for such a circumstance, we we'll use our old friend, the <div> tag.

```
<div class="main-container"></div>
```

We have just given <div> the class main-container which says that this is the main container for other item on our website.

Now to add the place for menu. It will be included in <div class="site-menu">:

```
<div class="main-container">
   <nav class="site-menu"></nav>
</div>
```

Moving forward, we place the container for the main content in the middle of the column:

```
<div class="main-container">
   <nav class="site-menu"></nav>
   <div class="main-content"></div>
</div>
```

And now the right column, or "sidebar":

```
<div class="main-container">
   <nav class="site-menu"></nav>
   <div class="main-content"></div>
   <aside class="sidebar"></aside>
</div>
```

Now let's analyze our next step based on the picture. What content should the left-column contain? As the name suggests, this should be our menu from the previous exercises:

```
<div class="main-container">
   <nav class="site-menu">
     insert menu code here
   </nav>
   <div class="main-content"></div>
   <aside class="sidebar"></aside>
</div>
```

I've put placeholder text inside of the <nav> tag for the menu code to preserve readability. When we are ready to publish our website, we will simply copy the code from our menu and paste it between <nav> and </nav>.

The middle column will contain the article and comment form:

```
<div class="main-container">
   <nav class="site-menu">
     insert menu code here
   </nav>
   <div class="main-content">
     <article>insert article code here</article>
     <form>insert form code here</form>
   </div>
   <aside class="sidebar"></aside>
</div>
```

For the right column, we'll put a side item unrelated to the main content, using the tag <aside>:

```
<div class="main-container">
  <nav class="site-menu">
    insert menu code here
  </nav>
  <div class="main-content">
    <article>insert article code here</article>
    <form>insert form code here</form>
  </div>
  <aside class="sidebar">
    <div>insert sidebar code here</div>
  </aside>
</div>
```

Ok, our HTML is ready. Now it's time for CSS. Our first task is to set the maximum width for the main container with class "main container":

```
.main-container {
  max-width: 960px;
}
```

We've given it the property max-width, so whatever happens, the width of the entire container named with a class main-container will never be wider than 960 pixels.

Next to center the block. This is done by setting automatic margins:

```
.main-container {
  max-width: 960px;
  margin: auto;
}
```

With this code, the browser will take up all free space around .main-container and distribute the space equally between the two margins.

auto space for left margin	div with margin: auto	auto space for right margin

We now have all the code responsible for the main container. Let's proceed to code our three columns. Let the container of the menu have 20% of the available width. This is done simply by specifying a percentage value:

```
.site-menu {
  width: 20%;
}
```

And a similar width for the right-hand column:

```
.sidebar {
  width: 20%;
}
```

Since the left and right column have the same properties, we can group them with a comma:

```
.site-menu,
.sidebar {
  width: 20%;
}
```

Now let's get to the middle column. It will take the remaining width (60%), as the two side columns occupy a total of 40%.

```
.main-content {
  width: 60%;
}
```

Unfortunately, our containers still appear in a block (one above the other). To set them next to each other, we need to give them a special CSS float property. We use it to tell the browser that we want X item closer to the left or right edge of the container in which it is placed. When one container has elements "floated" to the left

edge, they will set up next to each other and we can cleverly take advantage of this mechanic.

This is because the left-hand column "sticks" to the left edge of the main container.

Before we get to use CSS float, however, I want to show you the most common use case of this property. Imagine that you have an image set directly within text:

```
<p><img src="cat.jpg" alt="angry cat is watching you">Lorem
ipsum dolor sit amet, consectetur adipiscing elit. Ut dui metus,
commodo vitae sem vel, tempus pellentesque nunc. Cum sociis
natoque penatibus et magnis dis parturient montes, nascetur
ridiculus mus.  Etiam consequat, nisi ac cursus mattis, mi eros
lacinia tortor, nec pellentesque ligula quam mattis nulla.</p>
```

By default, it looks like that:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut dui metus, commodo vitae sem vel, tempus pellentesque nunc. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam consequat, nisi ac cursus mattis, mi eros lacinia tortor, nec pellentesque ligula quam mattis nulla.

This happens because the tag <img> is is a "line" component, so it was inserted at the beginning of the line and the text will simply follow it.

However, if we set the image to "float" to the "right" we will see something much better:

```
img {
   float: right;
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut dui metus, commodo vitae sem vel, tempus pellentesque nunc. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam consequat, nisi ac cursus mattis, mi eros lacinia tortor, nec pellentesque ligula quam mattis nulla.

The picture is now "floating" on the right edge of the text in which it was placed. In this case, the edges of the paragraph <p>.
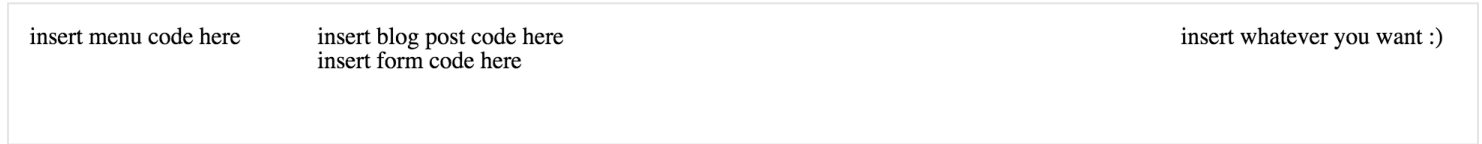
Now let's set float to left:

As you can see, the picture is now "floating" to the left edge of the paragraph, and the text wrapped around it on the right side.

Returning to our columns, we can do the same thing with to all columns that we want to position "left" at the edge of the container. Notice that we're about to float all containers as grouped with commas:

```
.site-menu,
.main-content,
.sidebar {
  float: left;
}
```

Bingo! See how our browser now displays the elements:

insert menu code here	insert blog post code here insert form code here	insert whatever you want :)

Let's analyze what happened after we "floated left." Each element will attempt to float left in terms of their priority in the list. So in this case, .site-nav took the leftmost edge, and .main-content was next in line to fill in the left-most space. Finally, the .sidebar filled in the last "left" edge, completing the arrangement.

Awesome! We've just obtained a multi-column arrangement. To further illustrate this topic, let's add background colors for each column:

```
.main-content {
  width: 60%;
  background-color: LightPink;
}

.site-menu {
  background-color: LightGoldenRodYellow;
}

.sidebar {
  background-color: LightSteelBlue;
}
```

The effect is satisfactory. The following result shows that we managed to divide the main container columns correctly:

insert menu code here	insert blog post code here insert form code here	insert whatever you want :)

Our final CSS code looks like this:

```css
.main-container {
  max-width: 960px;
  margin: auto;
}

.site-menu,
.main-content,
.sidebar {
  float: left;
}

.site-menu,
.sidebar {
  width: 20%;
}

.main-content {
  width: 60%;
  background-color: LightPink;
}

.site-menu {
  background-color: LightGoldenRodYellow;
}

.sidebar {
  background-color: LightSteelBlue;
}
```

With this code, we've created a simply layout with three columns. Now, you just need to paste the remaining HTML and CSS code from our previous exercises and our job is done!

You have completed your first website!

What's Next?

Congratulations on building your first full-fledged website. You skillfully created a page with three columns, a menu, and a comment form.

The next steps you perform as a developer are up to you, and there are endless possibilities. Perhaps you're wondering what you should do next? I have a few tips that might help!

First off, please subscribe to my newsletter. I will be sending you many helpful materials, including news about my other books which will be published shortly.

You can also follow me on Twitter to stay up to date with my work.

Then, choose your favorite website and try to play around and write the code from scratch. Find the sites' individual parts and identify their functionality. First create the HTML code, then apply the CSS (always in that order). After you finish, check it against the original and ask yourself what there is left to learn. That would be a perfect way to start!

PATTERNS

For the further development, it's crucial to realize that front-end programming (as a name for coding the user interfaces, in a web context it's mainly writing code in HTML, CSS and JavaScript) is a job based on well-known repeatable steps. Each time you work on a new website you will have to prepare a semantic HTML markup and

then write CSS code to get effects that reflect the website design you have been assigned to. The god is in the details though and that's why front-end is an interesting field.

As you learn more, you will quickly find out that there are usually many different ways to solve a problem and it's up to you to choose the most suitable. Even a simple circle can be displayed on the website in various ways. You can use an image, HTML5 canvas, SVG markup or CSS border-radius property. Remember our multi-column project? This can be also done in a few ways at least (try to Google for "inline-block layout", "flex-box column layout"). You need to figure out what fits your project needs best.

REALITY

The reality of the website programming is that modern browsers constantly improve by adding new CSS3 and HTML5 features. Week by week you get a browser update with tons of new features. As a developer, you need to incorporate all of them and leverage the most proper one based on different circumstances. You will have to answer questions like what browsers my projects aim to support and so on. You will have to decide if you care about a certain number of people using "X" browser. Then, more questions appear. "X" browser might be supporting rounded corners in CSS, while others not. If you wanted to support both browsers, you'd need to find a suitable workaround for this problem. It's one of the front-end developer's responsibilities – you will be selecting tools and techniques that will work in every browser your project is supposed to support.

HISTORY

Thus, it's important to learn a history of HTML5 and CSS3 development. It's been a crazy ride over last twenty years and some of the side-effects of this are still present. We have been constantly learning new things and while we have been trying to find answers, we have introduced new questions as well. You will be learning about them while approaching these problems. It's what the process of self-development looks like.

MAKE MISTAKES

Another important realization in order to become a good front-end programmer is that we all make mistakes and learn by them. There is no other way to improve yourself as a website developer than to constantly trying different things and learn from them. You need to be curious. The learning process is not measured in a finished time. Our industry moves fast and each month we have to get to know something new.

GOOD NEWS

However, the good news is that even though we are bombarded by a lot of new things each day, all of them are built on the foundations known for years. HTML and CSS don't drastically change their forms or a way you write in them. No. These languages remain more or less the same. Year after year we just have more answers to certain problems.

You might ask a question though. What is trending right now? What I need to be aware of?

ACCESSIBILITY

> 285 million people are estimated to be visually impaired worldwide: 39 million are blind and 246 have low vision. WHO

Many new concepts have come alive in recent years that will continue to grow in importance into the future. One of them is web accessibility.

> Web accessibility refers to the inclusive practice of removing barriers that prevent interaction with, or access to websites, by people with disabilities. When sites are correctly designed, developed and edited, all users have equal access to information and functionality.

A responsible developer just can't forget about it. I trully recommend you Web Accessibility: Web Standards and Regulatory Compliance and Apps For All: Coding Accessible Web Applications to get started on this important subject.

RESPONSIVE WEB DESIGN

Another recent web development technique is Responsive Web Design. In short, it's about how we design websites to appear nicely and readable on different devices including mobile phones, tablets and TVs. It actually makes sense to think a bit over why it makes a difference. In order to deliver a successful user experience (for example in your e-commerce business), you can't provide the same website layout for mobile phones and laptops. Well, actually you can, but it might turn out ineffective. Having a limited screen space on mobile, it might make sense to reconsider the website layout and accomodate the available space differently. And that's where RWD becomes handful. Thanks to it, you can set a specific bunch of CSS rules only for a certain screen resolutions so your website will look different on mobile devices whereas on laptops it can remain the same. Here's a simple example for setting a specific CSS rules only for devices wider than 320px:

```
@media screen and  (min-width: 320px) {
    .news {
        display: none;
    }
}
```

We used "media queries" here. In above snippet, every .news element in your code will be hidden for screen resolutions wider than 320px (because we set display: none).

There are of course more trending topics you will learn soon as you continue learning. In 2015, as web developers we build parallax websites, tell interactive stories using HTML5 video and audio or let people draw literally in the browser. Take a look at the following list to get a better sense of what modern web development is about:

- Species in Pieces – is an interactive exhibition turned study into 30 of the world's most interesting but unfortunately endangered species — their survivals laying literally, in pieces. All species are presented as CSS polygons.

- Apple MacBook – interesting animation of folding in/out the Macbook using HTML5 video

- WarsawRising.eu – a breathtaking story-telling project about Warsaw Uprising

- CSS Sans – fonts made using only CSS? No problem!

- CSS FPS – 3D graphics generated by CSS

- 50 problems in 50 days – Peter Smart attempted to solve 50 problems in 50 days using design and presented his amazing journey on a website.

- California Population Density

- Echoes of Tsunami – a terrific story of 2004's Tsunami and its effects. 10 years after the catastrophe, Action against Hunger commemorates the tsunami.

- Interactive Resume

- Financial Times App

- Every Last Drop

- Bezier game – learn the bezier curves by drawing them in a browser

Try to read blogs about HTML and CSS. Here's a list of valuable ones:

- Tim Berners-Lee, HTML creator
- Lea Verou
- Mary Rose Cook
- Harry Roberts
- Jeremy Keith
- Tantek Çelik
- Addy Osmani
- Christian Heilmann
- Jake Archibald
- Léonie Watson
- Sara Soueidan
- Todd Motto
- Kyle Simpson
- Rachel Nabors
- Peter Michaux
- Gunnar Bittersmann
- Frances Berriman
- Rodney Rehm
- Mathias Bynens
- Alicia Liu
- Alex Russell
- Remy Sharp
- Rachel Andrew

Try more new concepts. The following are very attractive topics:

- box model

- responsive web design

- progressive enhancement

- mobile first

- grid system

- css frameworks

- semantic web

- accessibility

- WAI-ARIA

Read good books:

- Dive into HTML5

- HTML5 for Web Designers

- CSS3 for Web Designers

- Illustrated Guide to Front-end Development

- JS For Cats

Read interesting articles and subscribe to valuable newsletters:

- CSS Tricks

- Smashing Magazine

- WebPlatform Daily

- JavaScript Weekly

Check also:

- Up to date with front-end technologies

Other than that, try to constantly learn new HTML5 tags and play with new things around CSS3. If you are not sure whether a feature you want to use is widely supported, go to Caniuse.com and check. If you look for a decent documentation,

please visit Mozilla Developer Network. Create an account on Twitter and follow most prominent people. Sign up at Github, publish your code and what's more important, try to analyze others' code. Moreover, experiment at codepen.io. Check JSBin or JSFiddle, where you can write and get an auto preview of your work. No need to save files locally!

Don't stop learning! Try and create. Be open-minded and tolerant. We all live in Open Web after all!