TABLE OF CONTENT

INTRODUCTION

Coding has exploded in recent years, changing from something used in computer games and the occasional electronic device, to something which shapes the way that we live in the modern world.

Pretty much every device, electronic item, and modern piece of machinery contains at least a little bit of code**.** As the number of use cases for coding grows, the number of coding jobs available will also continue to grow.

What is Coding?

Today, we are going to talk about coding. Specifically, I'll answer the question: *What in the world is coding?* We'll also cover a little of what happens when we code.

Before you read on, though, I want you to right-click in your browser window and choose the "View Page Source" option.

When you do that, you get a view of the web page code. Which is to say that, by peeking behind the curtain, you've now seen the language that tells your computer how to make this web page look as good as it does.

The first time I actually looked at the code in a browser window was revelatory for me. Here was the internet in the internet's own voice! I didn't understand most of it, but I could pick out snippets of words and phrases that looked familiar. There were a few font names that I recognized, and I understood pixel sizes more or less. The rest of it was alien to me. Here was the language my computer spoke, or so I thought.

If you're planning on learning to code, it's worth thinking through the mechanics of coding. Knowing what exactly is happening when you code, what it means when we say someone is coding, what the difference is between coding and programming, what languages you may end up coding in, and how to get started coding, will help you be a better coder.

Computer Coding for Beginners

There's a lot of hype around coding, so let's start by clarifying what coding isn't. I mentioned above that when I first saw the source code of a web page, I thought that I was looking at the language my computer spoke. This is a common way of explaining what code is, but it's not exactly true.

Your computer doesn't understand the nuances of language. In fact, the only terms your computer understands very well at all are "Yes" or "No."

Imagine you are building a bridge with a group of engineers. You're on one side, they are on the other, and you need to communicate to finish the project. The problem is, your phone has died, and your radio only works one way. All you have to communicate with them is a flashlight. One flash for yes, two for no. It will take a while but, eventually, the bridge will be built.

This is how a computer communicates with people. The language the computer speaks is binary code, a mathematical language of ones and zeros. Just like the flashlight, there are only two options. The computer understands "on" and "off," and nothing else. So unless you're typing strings of ones and zeros into your text editor (which you're not), you're not really writing code in the computer's language.

But if the code isn't written in the computer's language, what are you doing?

Computer Code is a Language

Think about writing code like this. You don't speak binary, and the machine can't come close to understanding human languages. So, for you to tell the computer what to do, you need to design a translator that can act as an intermediary. This is the purpose of code. Code is a form of writing that isn't binary, that is easy to learn and interpret for humans, but that the computer can still understand.

For most of the programs you're likely to work on, the code you write is actually a step removed from the binary code that the computer will process. You'll write in a code that pulls from human language. Programs built into your computer then translate what you've written into binary. It's like if you needed to speak to someone who in Mandarin, you only know English, and the only translator you could find spoke only Mandarin and French. You would need another translator to translate from English to French and then the first translator can translate French to Mandarin, hopefully without meaning getting lost in the process.

What sort of blows my mind about all of this is that it somehow works. We have programs translating programs for a machine that only speaks binary. This is an insanely complicated process, yet here I am typing human words on my binary speaking computer.

There's a lot more to it, of course, but these are the essential things to know before you start a conversation with your computer through code.

Coding vs. Programming: What's the Difference?

When I was growing up, my dad and all the people he worked with were computer programmers. This framed how I understood people who wrote code for computers for a long time: They were programmers.

More recently it seemed like there was a shift in either the terminology or the industry. Suddenly, people who wrote for computers were coders. This shift has prompted me

to wonder whether there is any difference between what programmers do and what coders do.

Many say that there really isn't a difference between a coder and a programmer at all. It's a difference in terminology rather than activity. If anything, in my humble opinion, saying you are a *coder* is slightly more general than saying you are a *programmer*. Since to me the word programmer is often associated with computer programming (and computer science) courses at a university. Whereas, coding feels like something everyone can do.

Jonah Bitautas, the product designer behind Cards Against Humanity, makes an interesting point when he argues that there is a real difference and it's rooted in issues of scale. Essentially, a coder is someone who writes language for computers. A programmer is someone who oversees the writing of a whole program — that is to say, an entire project's worth of coding.

At the end of the day, there is no formal definition of a coder or a programmer. And so, it's completely fine to use these terms interchangeably: you can say "I'm a coder" or "I'm a programmer," and no one will judge you for it. Unless during the rare occasion on Hacker News — the programming news website where seasoned (and opinionated) coders hang out. Sure, they might say something. Just brush it off and keep on moving on.

Types of Coding Languages

There are dozens of coding languages. A few languages are all-purpose (or multipurpose), but most serve a specific function. CSS, for example, primarily functions to make things look pretty. JavaScript, a relatively old language, exists to make web pages more functional. There are specialized languages that are great if you need something super specific, but all you really need to get started are a few common ones.

HTML

When I asked you to open the source code for this web page, it took you to lines of code written in HTML. Short for Hypertext Markup Language, HTML serves as the bones of the Internet. It tells web pages what should be displayed and where and how they'll fit within a given style sheet. It also tells your browser where to look for content like images and videos that you may want to include in your project, as well as where to find the style sheet you're working off of.

One thing to be aware of: HTML technically isn't a "programming language" because it doesn't use logic based expressions like, let's say, Python does. HTML is a markup language — but much like the debate of coder vs. programer vs. developer — you're unlikely to be faulted for calling it a programming language, especially if you are new.

CSS

CSS is the stylesheet. If you open up a CSS file, you'll see a lot of references to font families, colors, and font formatting styles (like bold, underlined, or italics). When your browser loads a page, the HTML tells it "Make this part of the page look like a header. OK?" It also says "Here's where to look to understand what a header should look like." This will always be a CSS file.

Javascript

Javascript is the language that brings interactivity to a web page. When you click a button on a web site, for example, it's JavaScript that makes the button look like you're clicking it. The controls for video players on the web and animations are also often Javascript.

Ruby vs. Python

We actually already have a baller post on the differences between Ruby and Python, but the short version is that these two programming languages are often used to develop web applications.

Which is to say, they create programs that allow web pages to do things at a high level of interactivity. If you want to, for example, build a bot to create an automatic payment system for your clients, you'll probably use one of these. They're great programs to learn to work with because they are extraordinarily versatile and there is a lot of extant code for you on the web to begin playing with this.

HTML

HTML (HyperText Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. Other technologies besides HTML are generally used to describe a web page's appearance/presentation (CSS) or functionality/behavior (JavaScript). "Hypertext" refers to links that connect web pages to one another, either within a single website or between websites. Links are a fundamental aspect of the Web. By uploading content to the Internet and linking it to pages created by other people, you become an active participant in the World Wide Web.

HTML uses "markup" to annotate text, images, and other content for display in a Web browser. HTML markup includes special "elements" such as `<head>`, `<title>`, `<body>`, `<header>`, `<footer>`, `<article>`, `<section>`, `<p>`, `<div>`, `<span>`, `<img>`, `<aside>`, `<audio>`, `<canvas>`, `<datalist>`, `<details>`, `<embed>`, `<nav>`, `<output>`, `<progress>`, `<video>`, `<ul>`, `<ol>`, `<li>` and many others.
An HTML element is set off from other text in a document by "tags", which consist of the element name surrounded by "<" and ">". The name of an element inside a tag is case insensitive. That is, it can be written in uppercase, lowercase, or a mixture. For example, the `<title>` tag can be written as `<Title>`, `<TITLE>`, or in any other way.

HTML is not a programming language; it is a *markup language* that defines the structure of your content. HTML consists of a series of **elements**, which you use to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way. The enclosing tags can make a word or image hyperlink to somewhere else, can italicize words, can make the font bigger or smaller, and so on. For example, take the following line of content:

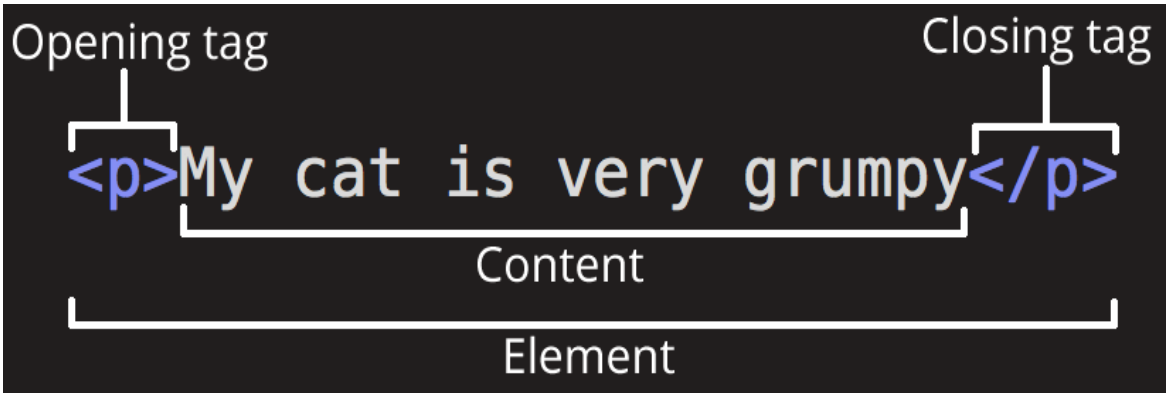
My cat is very grumpy
```
My cat is very grump
```

If we wanted the line to stand by itself, we could specify that it is a paragraph by enclosing it in paragraph tags:

My cat is very grumpy

Anatomy of an HTML element

Let's explore this paragraph element a bit further.



The main parts of our element are as follows:

1. **The opening tag:** This consists of the name of the element (in this case, p), wrapped in opening and closing **angle brackets**. This states where the element begins or starts to take effect — in this case where the paragraph begins.
2. **The closing tag:** This is the same as the opening tag, except that it includes a *forward slash* before the element name. This states where the element ends — in this case where the paragraph ends. Failing to add a closing tag is one of the standard beginner errors and can lead to strange results.
3. **The content:** This is the content of the element, which in this case, is just text.
4. **The element:** The opening tag, the closing tag, and the content together comprise the element.

Elements can also have attributes that look like the following:

Attributes contain extra information about the element that you don't want to appear in the actual content. Here, `class` is the attribute *name* and `editor-note` is the attribute *value*.
The `class` attribute allows you to give the element a non-unique identifier that can be used to target it (and any other elements with the same `class` value) with style information and other things.
An attribute should always have the following:

1. A space between it and the element name (or the previous attribute, if the element already has one or more attributes).
2. The attribute name followed by an equal sign.
3. The attribute value wrapped by opening and closing quotation marks.

Note: Simple attribute values that don't contain ASCII whitespace (or any of the characters `" ' ` = < >`) can remain unquoted, but it is recommended that you quote all attribute values, as it makes the code more consistent and understandable.

Nesting elements

You can put elements inside other elements too — this is called **nesting**. If we wanted to state that our cat is **very** grumpy, we could wrap the word "very" in a `<strong>` element, which means that the word is to be strongly emphasized:

```
<p>My cat is <strong>very</strong> grumpy.</p>
```

You do however need to make sure that your elements are properly nested. In the example above, we opened the `<p>` element first, then the `<strong>` element; therefore, we have to close the `<strong>` element first, then the `<p>` element. The following is incorrect:

```
<p>My cat is <strong>very grumpy.</p></strong>
```

The elements have to open and close correctly so that they are clearly inside or outside one another. If they overlap as shown above, then your web browser will try to make the best guess at what you were trying to say, which can lead to unexpected results. So don't do it!

Empty elements

Some elements have no content and are called **empty elements**. Take the `<img>` element that we already have in our HTML page:

```
<img src="images/firefox-icon.png" alt="My test image">
```

This contains two attributes, but there is no closing `</img>` tag and no inner content. This is because an image element doesn't wrap content to affect it. Its purpose is to embed an image in the HTML page in the place it appears.

Anatomy of an HTML document

That wraps up the basics of individual HTML elements, but they aren't handy on their own. Now we'll look at how individual elements are combined to form an entire HTML page. Let's revisit the code we put into our `index.html` example (which we first met in the Dealing with files article):

```
<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8">

    <title>My test page</title>

  </head>
```

```
  <body>

    <img src="images/firefox-icon.png" alt="My test image">

  </body>

</html>
```

Here, we have the following:

- `<!DOCTYPE html>` — doctype. It is a required preamble. In the mists of time, when HTML was young (around 1991/92), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML, which could mean automatic error checking and other useful things. However these days, they don't do much and are basically just needed to make sure your document behaves correctly. That's all you need to know for now.
- `<html></html>` — the `<html>` element. This element wraps all the content on the entire page and is sometimes known as the root element.
- `<head></head>` — the `<head>` element. This element acts as a container for all the stuff you want to include on the HTML page that *isn't* the content you are showing to your page's viewers. This includes things like keywords and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more.
- `<meta charset="utf-8">` — This element sets the character set your document should use to UTF-8 which includes most characters from the vast majority of written languages. Essentially, it can now handle any textual content you might put on it. There is no reason not to set this and it can help avoid some problems later on.
- `<title></title>` — the `<title>` element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in. It is also used to describe the page when you bookmark/favorite it.
- `<body></body>` — the `<body>` element. This contains *all* the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

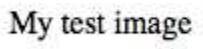
Images

Let's turn our attention to the `<img>` element again:

```
<img src="images/firefox-icon.png" alt="My test image">
```

As we said before, it embeds an image into our page in the position it appears. It does this via the `src` (source) attribute, which contains the path to our image file.
We have also included an `alt` (alternative) attribute. In this attribute, you specify descriptive text for users who cannot see the image, possibly because of the following reasons:

1. They are visually impaired. Users with significant visual impairments often use tools called screen readers to read out the alt text to them.
2. Something has gone wrong causing the image not to display. For example, try deliberately changing the path inside your `src` attribute to make it incorrect. If you save and reload the page, you should see something like this in place of the image:



The keywords for alt text are "descriptive text". The alt text you write should provide the reader with enough information to have a good idea of what the image conveys. In this example, our current text of "My test image" is no good at all. A much better alternative for our Firefox logo would be "The Firefox logo: a flaming fox surrounding the Earth."

Try coming up with some better alt text for your image now.

Find out more about accessibility in our accessibility learning module.

Marking up text

This section will cover some of the essential HTML elements you'll use for marking up the text.

Headings

Heading elements allow you to specify that certain parts of your content are headings — or subheadings. In the same way that a book has the main title, chapter titles, and subtitles, an HTML document can too. HTML contains 6 heading levels, `<h1>`–`<h6>`, although you'll commonly only use 3 to 4 at most:

```
<h1>My main title</h1>

    <h2>My top level heading</h2>

    <h3>My subheading</h3>

    <h4>My sub-subheading</h4>
```

Now try adding a suitable title to your HTML page just above your `<img>` element.

Note: You'll see that your heading level 1 has an implicit style. Don't use heading elements to make text bigger or bold, because they are used for accessibility and other reasons such as SEO. Try to create a meaningful sequence of headings on your pages, without skipping levels.

Paragraphs

As explained above, `<p>` elements are for containing paragraphs of text; you'll use these frequently when marking up regular text content:

```
<p>This is a single paragraph</p>
```

Add your sample text (you should have it from *What should your website look like?*) into one or a few paragraphs, placed directly below your `<img>` element.

Lists

A lot of the web's content is lists and HTML has special elements for these. Marking up lists always consists of at least 2 elements. The most common list types are ordered and unordered lists:

1. **Unordered lists** are for lists where the order of the items doesn't matter, such as a shopping list. These are wrapped in a `<ul>` element.
2. **Ordered lists** are for lists where the order of the items does matter, such as a recipe. These are wrapped in an `<ol>` element.
   Each item inside the lists is put inside an `<li>` (list item) element.
   For example, if we wanted to turn the part of the following paragraph fragment into a list

```
<p>At Mozilla, we're a global community of technologists, thinkers, and builders working together ... </p>
```

We could modify the markup to this

```
<p>At Mozilla, we're a global community of</p>



<ul>

  <li>technologists</li>

  <li>thinkers</li>

  <li>builders</li>
```

```
</ul>

<p>working together ... </p>
```

Try adding an ordered or unordered list to your example page.

Links

Links are very important — they are what makes the web a web! To add a link, we need to use a simple element — `<a>` — "a" being the short form for "anchor". To make text within your paragraph into a link, follow these steps:

1. Choose some text. We chose the text "Mozilla Manifesto".
2. Wrap the text in an `<a>` element, as shown below:

   ```
   <a>Mozilla Manifesto</a>
   ```

3. Give the `<a>` element an `href` attribute, as shown below:

   ```
   <a href="">Mozilla Manifesto</a>
   ```

4. Fill in the value of this attribute with the web address that you want the link to link to:

   ```
   <a href="https://www.mozilla.org/en-US/about/manifesto/">Mozilla Manifesto</a>
   ```

You might get unexpected results if you omit the `https://` or `http://` part, called the *protocol*, at the beginning of the web address. After making a link, click it to make sure it is sending you where you wanted it to.

`href` might appear like a rather obscure choice for an attribute name at first. If you are having trouble remembering it, remember that it stands for ***h****ypertext* ***ref****erence*.

Add a link to your page now, if you haven't already done so.

CSS

CSS is the stylesheet. If you open up a CSS file, you'll see a lot of references to font families, colors, and font formatting styles (like bold, underlined, or italics). When your browser loads a page, the HTML tells it "Make this part of the page look like a header. OK?" It also says "Here's where to look to understand what a header should look like." This will always be a CSS file.

Including CSS in HTML Documents

CSS can either be attached as a separate document or embedded in the HTML document itself. There are three methods of including CSS in an HTML document:

- **Inline styles** — Using the `style` attribute in the HTML start tag.
- **Embedded styles** — Using the `<style>` element in the head section of a document.
- **External style sheets** — Using the `<link>` element, pointing to an external CSS file.

In this tutorial we will cover all these three methods for inserting CSS one by one.

Note: The inline styles have the highest priority, and the external style sheets have the lowest. It means if you specify styles for an element in both *embedded* and *external* style sheets, the conflicting style rules in the embedded style sheet would override the external style sheet.

Inline Styles

Inline styles are used to apply the unique style rules to an element by putting the CSS rules directly into the start tag. It can be attached to an element using the `style` attribute.

The `style` attribute includes a series of CSS property and value pairs. Each `"property: value"` pair is separated by a semicolon (`;`), just as you would write into an embedded or external style sheets. But it needs to be all in one line i.e. no line break after the semicolon, as shown here:

Example
Try this code »

```
<h1 style="color:red; font-size:30px;">This is a heading</h1>

<p style="color:green; font-size:22px;">This is a paragraph.</p>

<div style="color:blue; font-size:14px;">This is some text content.</div>
```

Using the inline styles are generally considered as a bad practice. As style rules are embedded directly inside the HTML tag, it causes the presentation to become mixed with the content of the document; which makes the code hard to maintain and negates the purpose of using CSS.

Note: It's become impossible to style pseudo-elements and pseudo-classes with inline styles. You should, therefore, avoid the use of style attributes in your code.
Using external style sheets is the preferred way to add styles to the HTML documents.

Embedded Style Sheets

Embedded or internal style sheets only affect the document they are embedded in.

Embedded style sheets are defined in the `<head>` section of an HTML document using the `<style>` element. You can define any number of `<style>` elements in an HTML document but they must appear between the `<head>` and `</head>` tags. Let's take a look at an example:

Example
Try this code »

```
<!DOCTYPE html>

<html lang="en">

<head>

   <title>My HTML Document</title>

   <style>

      body { background-color: YellowGreen; }

      p { color: #fff; }
```

```
    </style>
</head>
<body>
    <h1>This is a heading</h1>
    <p>This is a paragraph of text.</p>
</body>
</html>
```

Tip: The `type` attribute of the `<style>` and `<link>` tag (i.e. `type="text/css"`) defines the language of the style sheet. This attribute is purely informative. You can omit this since CSS is the standard and default style sheet language in HTML5.

External Style Sheets

An external style sheet is ideal when the style is applied to many pages of the website.

An external style sheet holds all the style rules in a separate document that you can link from any HTML file on your site. External style sheets are the most flexible because with an external style sheet, you can change the look of an entire website by changing just one file.

You can attach external style sheets in two ways — *linking* and *importing*.

Linking External Style Sheets

Before linking, we need to create a style sheet first. Let's open your favorite code editor and create a new file. Now type the following CSS code inside this file and save it as "style.css".

Example

Try this code »

```
body {
    background: lightyellow;
    font: 18px Arial, sans-serif;
}
h1 {color: orange;}
```

An external style sheet can be linked to an HTML document using the `<link>` tag.
The `<link>` tag goes inside the `<head>` section, as you can see in the following example:

Example
Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My HTML Document</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <h1>This is a heading</h1>
    <p>This is a paragraph of text.</p>
</body>
</html>
```

Tip: Among all the three methods, using external style sheet is the best method for defining and applying styles to the HTML documents. As you can clearly see with external style sheets, the affected HTML file require minimal changes in the markup.

Importing External Style Sheets

The `@import` rule is another way of loading an external style sheet. The `@import` statement instructs the browser to load an external style sheet and use its styles.

You can use it in two ways. The simplest is within the header of your document. Note that, other CSS rules may still be included in the `<style>` element. Here's an example:

Example
Try this code »

```
<style>
    @import url("css/style.css");
    p {
        color: blue;
        font-size: 16px;
    }
</style>
```

Similarly, you can use the `@import` rule to import a style sheet within another style sheet.

Example
Try this code »

```
@import url("css/layout.css");
@import url("css/color.css");
body {
    color: blue;
```

```
    font-size: 14px;

}
```

Note: All `@import` rules must occur at the start of the style sheet. Any style rule defined in the style sheet itself override the conflicting rules in the imported style sheets. However, importing a style sheet within another style sheet is not recommended due to performance issue.

Understanding CSS Syntax

A CSS stylesheet consists of a set of rules that are interpreted by the web browser and then applied to the corresponding elements such as paragraphs, headings, etc. in the document.

A CSS rule have two main parts, a selector and one or more declarations:

The selector specifies which element or elements in the HTML page the CSS rule applies to.

Whereas, the declarations within the block determines how the elements are formatted on a webpage. Each declaration consists of a property and a value separated by a colon (`:`) and ending with a semicolon (`;`), and the declaration groups are surrounded by curly braces `{}`.

The property is the style attribute you want to change; they could be font, color, background, etc. Each property has a value, for example color property can have value either `blue` or `#0000FF` etc.

```
h1 {color:blue; text-align:center;}
```

To make the CSS more readable, you can put one declaration on each line, like this:

Example
Try this code »

```
h1 {
    color: blue;
```

```
    text-align: center;
}
```

In the example above `h1` is a selector, `color` and `text-align` are the CSS properties, and the given `blue` and `center` are the corresponding values of these properties.

Note: A CSS declaration always ends with a semicolon ";", and the declaration groups are always surrounded by the curly brackets "`{}`".

Writing Comments in CSS

Comments are usually added with the purpose of making the source code easier to understand. It may help other developer (or you in the future when you edit the source code) to understand what you were trying to do with the CSS. Comments are significant to programmers but ignored by browsers.

A CSS comment begins with `/*`, and ends with `*/`, as shown in the example below:

Example
Try this code »

```
/* This is a CSS comment */
h1 {
    color: blue;
    text-align: center;
}
/* This is a multi-line CSS comment
that spans across more than one line */
p {
```

```
    font-size: 18px;

    text-transform: uppercase;

}
```

You can also comment out part of your CSS code for debugging purpose, as shown here:

Example
Try this code »

```
h1 {

    color: blue;

    text-align: center;

}

/*

p {

    font-size: 18px;

    text-transform: uppercase;

}

*/
```

Case Sensitivity in CSS

CSS property names and many values are not case-sensitive. Whereas, CSS selectors are usually case-sensitive, for instance, the class selector `.maincontent` is not the same as `.mainContent`.

Therefore, to be on safer side, you should assume that all components of CSS rules are case-sensitive.

CSS Selectors

What is Selector?

A CSS selector is a pattern to match the elements on a web page. The style rules associated with that selector will be applied to the elements that match the selector pattern.

Selectors are one of the most important aspects of CSS as they allow you to target specific elements on your web page in various ways so that they can be styled.

Several types of selectors are available in CSS, let's take a closer look at them:

Universal Selector

The universal selector, denoted by an asterisk (`*`), matches every single element on the page.

The universal selector may be omitted if other conditions exist on the element. This selector is often used to remove the default margins and paddings from the elements for quick testing purpose.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
* {
    margin: 0;
    padding: 0;
}
```

The style rules inside the `*` selector will be applied to every element in a document.

Note: It is recommended *not to use* the universal selector (`*`) too often in a production environment, since this selector matches every element on a web page that puts too

much of unnecessary pressure on the browsers. Use element type or class selector instead.

Element Type Selectors

An element type selector matches all instance of the element in the document with the corresponding element type name. Let's try out an example to see how it actually works:

Example
Try this code »

```
p {
    color: blue;
}
```

The style rules inside the `p` selector will be applied on every `<p>` element (or paragraph) in the document and color it blue, regardless of their position in the document tree.

Id Selectors

The id selector is used to define style rules for a *single* or *unique* element.

The id selector is defined with a hash sign (`#`) immediately followed by the id value.

Example
Try this code »

```
#error {
    color: red;
}
```

This style rule renders the text of an element in red, whose `id` attribute is set to `error`.

Note: The value of an id attribute must be unique within a given document — meaning no two elements in your HTML document can share the same id value.

Class Selectors

The class selectors can be used to select any HTML element that has a `class` attribute. All the elements having that class will be formatted according to the defined rule.

The class selector is defined with a period sign (`.`) immediately followed by the class value.

Example
Try this code »S

```
.blue {
    color: blue;
}
```

The above style rules renders the text in blue of every element in the document that has `class` attribute set to `blue`. You can make it a bit more particular. For example:

Example
Try this code »

```
p.blue {
    color: blue;
}
```

The style rule inside the selector `p.blue` renders the text in blue of only those `<p>` elements that has `class` attribute set to `blue`, and has no effect on other paragraphs.

Descendant Selectors

You can use these selectors when you need to select an element that is the descendant of another element, for example, if you want to target only those anchors that are contained within an unordered list, rather than targeting all anchor elements. Let's see how it works:

Example
Try this code »

```
ul.menu li a {

    text-decoration: none;

}

h1 em {

    color: green;

}
```

The style rules inside the selector `ul.menu li a` applied to only those `<a>` elements that contained inside an `<ul>` element having the class `.menu`, and has no effect on other links inside the document.

Similarly, the style rules inside the `h1 em` selector will be applied to only those `<em>` elements that contained inside the `<h1>` element and has not effect on other `<em>` elements.

Child Selectors

A child selector is used to select only those elements that are the direct children of some element.

A child selector is made up of two or more selectors separated by a greater than symbol (`>`). You can use this selector, for instance, to select the first level of list elements inside a nested list that has more than one level. Let's check out an example to understand how it works:

Example
Try this code »

```
ul > li {

    list-style: square;

}

ul > li ol {

    list-style: none;

}
```

The style rule inside the selector `ul > li` applied to only those `<li>` elements that are direct children of the `<ul>` elements, and has no effect on other list elements.

Adjacent Sibling Selectors

The adjacent sibling selectors can be used to select sibling elements (i.e. elements at the same level). This selector has the syntax like: E1 + E2, where E2 is the target of the selector.

The selector `h1 + p` in the following example will select the `<p>` elements only if both the `<h1>` and `<p>` elements share the same parent in the document tree and `<h1>` is immediately precedes the `<p>` element. That means only those paragraphs that come immediately after each `<h1>` heading will have the associated style rules. Let's see how this selector actually works:

Example
Try this code »

```
h1 + p {

   color: blue;
```

```
  font-size: 18px;
}
ul.task + p {
  color: #f0f;
  text-indent: 30px;
```

General Sibling Selectors

The general sibling selector is similar to the adjacent sibling selector (E1 + E2), but it is less strict. A general sibling selector is made up of two simple selectors separated by the tilde (~) character. It can be written like: E1 ~ E2, where E2 is the target of the selector.

The selector `h1 ~ p` in the example below will select all the `<p>` elements that preceded by the `<h1>` element, where all the elements share the same parent in the document tree.

Example
Try this code »

```
h1 ~ p {
  color: blue;
  font-size: 18px;
}
ul.task ~ p {
  color: #f0f;
```

```
    text-indent: 30px;
}
```

There are more sophisticated selectors like attribute selectors, pseudo-classes, pseudo-elements. We will discuss about these selectors in detail in the upcoming chapters.

Grouping Selectors

Often several selectors in a style sheet share the same style rules declarations. You can group them into a comma-separated list to minimize the code in your style sheet. It also prevents you from repeating the same style rules over and over again. Let's take a look:

Example
Try this code »

```
h1 {
    font-size: 36px;
    font-weight: normal;
}
h2 {
    font-size: 28px;
    font-weight: normal;
}
h3 {
    font-size: 22px;
    font-weight: normal;
```

}

As you can see in the above example, the same style rule `font-weight: normal;` is shared by the selectors `h1`, `h2` and `h3`, so it can be grouped in a comma-separated list, like this:

Example
Try this code »

```
h1, h2, h3 {
  font-weight: normal;
}
h1 {
  font-size: 36px;
}
h2 {
  font-size: 28px;
}
h3 {
  font-size: 22px;
```

CSS Color

Setting Color Property

The `color` property defines the text color (foreground color in general) of an element.

For instance, the `color` property specified in the `body` selector defines the default text color for the whole page. Let's try out the following example to see how it works:

Example
Try this code »

```
body {
```

```
    color: #ff5722;

}
```

Note: The `color` property normally inherits the color value from their parent element, except the case of anchor elements. For example, if you specify `color` for the `body` element it will automatically be passed down to the headings, paragraphs, etc.

Defining Color Values

Colors in CSS most often specified in the following formats:

- a color keyword - like "red", "green", "blue", "transparent", etc.
- a HEX value - like "#ff0000", "#00ff00", etc.
- an RGB value - like "rgb(255, 0, 0)"

CSS3 has introduced several other color formats such as HSL, HSLA and RGBA that also support alpha transparency. We'll learn about them in greater detail in CSS3 color chapter.

For now, let's stick to the basic methods of defining the color values:

Color Keywords

CSS defines the few color keywords which lets you specify color values in an easy way.

These basic color keywords are: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow. The color names are case-insensitive.

Example
Try this code »

```
h1 {

    color: red;

}
```

```
p {
    color: purple;
}
```

Modern web browsers however practically support many more color names than what are defined in the CSS standard, but to be on the safer side you should use hex color values instead.

See the reference on CSS color names, for a complete list of possible color names.

HEX Color Values

Hex (short for Hexadecimal) is by far the most commonly used method of defining color on the web.

Hex represent colors using a six-digit code, preceded by a hash character, like `#rrggbb`, in which `rr`, `gg`, and `bb` represents the red, green and blue component of the color respectively.

The value of each component can vary from 00 (no color) and FF (full color) in hexadecimal notation, or 0 and 255 in decimal equivalent notation.
Thus `#ffffff` represents white color and `#000000` represents black color. Let's take a look the following example:

Example
Try this code »

```
h1 {
    color: #ffa500;
}
p {
    color: #00ff00;
}
```

Note: Hexadecimal or Hex refers to a numbering scheme that uses 16 characters as its base. It uses the numbers 0 through 9 and the letters A, B, C, D, E and F which corresponds to the decimal numbers 10, 11, 12, 13, 14 and 15 respectively.

Tip: If hexadecimal code of a color has value pairs, it can also be written in shorthand notation to avoid extra typing, for example, the hex color value #ffffff can be also be written as #fff, #000000 as #000, #00ff00 as #0f0, #ffcc00 as #fc0, and so on.

RGB Color Values

Colors can be defined in the RGB model (Red, Green, and Blue) using the `rgb()` functional notation.

The `rgb()` function accepts three comma-separated values, which specify the amount of red, green, and blue component of the color. These values are commonly specified as integers between 0 to 255, where 0 represent *no color* and 255 represent *full or maximum color*.

The following example specifies the same color as in the previous example but in RGB notation.

Example
Try this code »

```
h1 {
    color: rgb(255, 165, 0);
}
p {
    color: rgb(0, 255, 0);
}
```

Note: You can also specify RGB values inside the `rgb()` function in percentage, where 100% represents full color, and 0% (*not simply 0*) represents no color. For example, you can specify the red color either as `rgb(255, 0, 0)` or `rgb(100%, 0%, 0%)`.

Tip: If R, G, and B are all set to 255, i.e. `rgb(255, 255, 255)`, the color would be white. Likewise, if all channels are set to 0, i.e. `rgb(0, 0, 0)`, the color would be black. Play with the RGB values in the following demonstration to understand how it actually works.

rgb(127, 127, 127)

Affect of Color Property on Borders and Outlines

The `color` property is not just for text content, but for anything in the foreground that takes a color value. For instance, if `border-color` or `outline-color` value hasn't been defined explicitly for the element, the color value will be used instead. Let's check out an example:

Example
Try this code »

```
p.one {
  color: #0000ff;
  border: 2px solid;
}
p.two {
  color: #00ff00;
  outline: 2px solid;
}
```

CSS Background

Setting Background Properties

Background plays an important role in the visual presentation of a web page.

CSS provide several properties for styling the background of an element, including coloring the background, placing images in the background and managing their positioning, etc.

The background properties are `background-color`, `background-image`, `background-repeat`, `background-attachment` and `background-position`.

In the following section we will discuss each of these properties in more detail.

Background Color

The `background-color` property is used to set the background color of an element.

The following example demonstrates how to set the background color of the whole page.

Example
Try this code »

```
body {
    background-color: #f0e68c;
}
```

Color values in CSS are most often specified in the following formats:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255, 0, 0)"

Please check out the tutorial on CSS color to learn more about specifying color values.

Background Image

The `background-image` property set an image as a background of an HTML element.

Let's check out the following example which sets the background image for the whole page.

Example
Try this code »

```
body {

    background-image: url("images/tile.png");

}
```

Note: When applying the background image to an element, make sure that the image you choose does not affect the readability of the element's text content.

Tip: By default browser repeats or tiles the background image both horizontally and vertically to fill the entire area of an element. You can control this with `background-repeat` property.

Background Repeat

The `background-repeat` property allows you to control how a background image is repeated or tiled in the background of an element. You can set a background image to repeat vertically (y-axis), horizontally (x-axis), in both directions, or in neither direction.

Let's try out the following example which demonstrates how to set the gradient background for a web page by repeating the sliced image horizontally along the x-axis.

Example
Try this code »

```
body {

    background-image: url("images/gradient.png");
```

```
    background-repeat: repeat-x;

}
```

Similarly, you can use the value `repeat-y` to repeat the background image vertically along the y-axis, or the value `no-repeat` to prevent the repetition altogether.

Example
Try this code »

```
body {

    background-image: url("images/texture.png");

    background-repeat: no-repeat;

}
```

Let's take a look at the following illustration to understand how this property actually works.

Background Position

The `background-position` property is used to control the position of the background image.

If no background position has been specified, the background image is placed at the default top-left position of the element i.e. at `(0,0)`, let's try out the following example:

Example
Try this code »

```
body {

    background-image: url("images/robot.png");

    background-repeat: no-repeat;

}
```

In the following example, the background image is positioned at top-right corner.

Example
Try this code »

```
body {
    background-image: url("images/robot.png");
    background-repeat: no-repeat;
    background-position: right top;
}
```

Note: If two values are specified for the `background-position` property, the first value represents the horizontal position, and the second represents the vertical position. If only one value is specified, the second value is assumed to be center.

Besides keywords, you can also use percentage or length values, such as `px` or `em` for this property.

Let's take a look at the following illustration to understand how this property actually works.

Background Attachment

The `background-attachment` property determines whether the background image is fixed with regard to the viewport or scrolls along with the containing block.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
body {
```

```
    background-image: url("images/bell.png");

    background-repeat: no-repeat;

    background-attachment: fixed;

}
```

The Background Shorthand Property

As you can see in the examples above, there are many properties to consider when dealing with the backgrounds. However, it is also possible to specify all these properties in one single property to shorten the code or avoid extra typing. This is called a shorthand property.

The `background` property is a shorthand property for setting all the individual background properties, i.e., `background-color`, `background-image`, `background-repeat`, `background-attachment` and the `background-position` property at once. Let's see how this works:

Example
Try this code »

```
body {

    background-color: #f0e68c;

    background-image: url("images/smiley.png");

    background-repeat: no-repeat;

    background-attachment: fixed;

    background-position: 250px 25px;

}
```

Using the shorthand notation the above example can be written as:

Example
Try this code »

```
body {
    background: #f0e68c url("images/smiley.png") no-repeat fixed 250px 25px;
}
```

When using the `background` shorthand property the order of the property values should be.

```
background: color image repeat attachment position;
```

If the value for an individual background property is missing or not specified while using the shorthand notation, the default value for that property will be used instead, if any.

Note: The background properties do not inherit like the color property, but the parent element's background will be visible through by default, because of the initial or default `transparent` value of the `background-color` CSS property.

CSS Fonts

Styling Fonts with CSS

Choosing the right font and style is very crucial for the readability of text on a page.

CSS provide several properties for styling the font of the text, including changing their face, controlling their size and boldness, managing variant, and so on.

The font properties are: `font-family`, `font-style`, `font-weight`, `font-size`, and `font-variant`.

Let's discuss each of these font properties one by one in more detail.

Font Family

The `font-family` property is used to specify the font to be used to render the text.

This property can hold several comma-separated font names as a *fallback* system, so that if the first font is not available on the user's system, browser tries to use the second one, and so on.

Hence, list the font that you want first, then any fonts that might fill in for the first if it is not available. You should end the list with a generic font family which are five — `serif`, `sans-serif`, `monospace`, `cursive` and `fantasy`. A typical font family declaration might look like this:

Example
Try this code »

```
body {

    font-family: Arial, Helvetica, sans-serif;

}
```

Note: If the name of a font family contains more than one word, it must be placed inside quotation marks, like `"Times New Roman"`, `"Courier New"`, `"Segoe UI"`, etc.

The most common font families used in web design are *serif* and *sans-serif*, because they are more suitable for reading. While *monospace* fonts are commonly used to display code, because in this typeface each letter takes up the same space which looks like typewritten text.

The *cursive* fonts look like cursive writing or handwriting. The *fantasy* font represents artistic font, but they're not used widely because of poor availability across the operating systems.

Difference Between Serif and Sans-serif Fonts

Serif fonts have small line or stroke at the extremities of characters, whereas sans-serif fonts are straighter and do not have these small strokes. See the following illustration.

Font Style

The `font-style` property is used to set the font face style for the text content of an element.

The font style can be `normal`, `italic` or `oblique`. The default value is `normal`.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
p.normal {
  font-style: normal;
}
p.italic {
  font-style: italic;
}
p.oblique {
  font-style: oblique;
}
```

Note: At first glance both oblique and italic font styles appear the same thing, but there is a difference. The `italic` style uses an *italic version* of the font while `oblique` style on the other hand is simply a slanted or sloped version of the normal font.

Font Size

The `font-size` property is used to set the size of font for the text content of an element.

There are several ways to specify the font size values e.g. with keywords, percentage, pixels, ems, etc.

Setting Font Size with Pixels

Setting the font size in pixel values (e.g. 14px, 16px, etc.) is a good choice when you need the pixel accuracy. Pixel is an absolute unit of measurement which specifies a fixed length.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
h1 {
    font-size: 24px;
}
p {
    font-size: 14px;
}
```

Defining the font sizes in pixel is not considered very accessible, because the user cannot change the font size from the browser settings. For instance, users with limited or low vision may wish to set the font size much larger than the size specified by you.

Therefore, you should avoid using the pixels values and use the values that are relative to the user's default font size instead if you want to create an inclusive design.

Tip: The text can also be resized in all browsers using the *zoom feature*. However, this feature resizes the entire page, not just the text. The W3C recommends using the em or percentage (%) values in order to create more robust and scalable layouts.

Setting Font Size with EM

The `em` unit refers to the font size of the parent element. When defining the `font-size` property, `1em` is equal to the size of the font that applies to the *parent of the element*.

So, if you set a `font-size` of 20px on the body element,
then `1em = 20px` and `2em = 40px`.

However, if you haven't set the font size anywhere on the page, then it is the browser default, which is normally 16px. Therefore, by default `1em = 16px`, and `2em = 32px`.

Let's take a look at the following example to understand how it basically works:

Example
Try this code »

```
h1 {

    font-size: 2em;    /* 32px/16px=2em */

}

p {

    font-size: 0.875em;    /* 14px/16px=0.875em */

}
```

Using the Combination of Percentage and EM

As you've observed in the above example the calculation of em values doesn't look straightforward. To simplify this, a popular technique is to set the `font-size` for the body element to `62.5%` (that is 62.5% of the default 16px), which equates to 10px, or 0.625em.

Now you can set the `font-size` for any elements using em units, with an easy-to-remember conversion, by dividing the `px` value by 10. This
way `10px = 1em`, `12px = 1.2em`, `14px = 1.4em`, `16px = 1.6em`, and so on. Let's take a look at the following example:

Example
Try this code »

```
body {
    font-size: 62.5%;    /* font-size 1em = 10px */
}
p {
    font-size: 1.4em;    /* 1.4em = 14px */
}
p span {
    font-size: 2em;    /* 2em = 28px */
}
```

Setting Font Size with Root EM

To make things even more simpler CSS3 has introduced `rem` unit (short for "root em") which is always relative to the font-size of the root element (`html`), regardless of where the element lies in the document (unlike `em` which is relative to parent element's font size).

This means that `1rem` is equivalent to the font size of the `html` element, which is `16px` by default in most browsers. Let's try out an example to understand how it actually works:

Example
Try this code »

```
html {
    font-size: 62.5%;    /* font-size 1em = 10px */
}
p {
    font-size: 1.4rem;    /* 1.4rem = 14px */
```

```
}

p span {

    font-size: 2rem;    /* 2rem = 20px (not 28px) */

}
```

Setting Font Size with Keywords

CSS provide several keywords that you can use to define font sizes.

An absolute font size can be specified using one of the following keywords: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`. Whereas, a relative font size can be specified using the keywords: `smaller` or `larger`. Let's try out an example and see how it works:

Example
Try this code »

```
body {

    font-size: large;

}

h1 {

    font-size: larger;

}

p {

    font-size: smaller;

}
```

Note: The keyword `medium` is equivalent to the browsers default font-size, which is normally 16px. Likewise, xx-small is the equivalent of 9 pixels, x-small is 10 pixels, small is 13 pixels, large is 18 pixels, x-large is 24 pixels, and xx-large is 32 pixels.

Tip: By setting a font size on the body element, you can set the relative font sizing everywhere else on the page, giving you the ability to easily scale the font size up or down accordingly.

Setting Font Size with Viewport Units

The font sizes can be specified using viewport units such as `vw` or `vh`.

Viewport units refer to a percentage of the browser's viewport dimensions, where 1vw = 1% of viewport width, and 1vh = 1% of viewport height. Hence, if the viewport is 1600px wide, 1vw is 16px.

Try out the following example by resizing the browser window and see how it works:

Example
Try this code »

```
body {
    font-size: 1vw;
}
h1 {
    font-size: 3vw;
}
```

However, there is a problem with the viewport units. On small screens fonts become so small that they are hardly readable. To prevent this you can utilize CSS `calc()` function, like this:

Example
Try this code »

```
html {
    font-size: calc(1em + 1vw);
```

```
}

h1 {

    font-size: 3rem;

}
```

In this example, even if the viewport width becomes 0, the font-size would be at least 1em or 16px.

You further utilize CSS media queries to create better responsive and fluid typography.

Font Weight

The `font-weight` property specifies the weight or boldness of the font.

This property can take one of the following values: `normal`, `bold`, `bolder`, `lighter`, `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, `900` and `inherit`.

- The numeric values `100`-`900` specify the font weights, where each number represents a weight greater than its predecessor. `400` is same as `normal` & `700` is same as `bold`.
- The `bolder` and `lighter` values are relative to the inherited font weight, while the other values such as `normal` and `bold` are absolute font weights.

Let's try out an example to understand how this property basically works:

Example
Try this code »

```
p {

    font-weight: bold;
```

Note: Most of the fonts are only available in a limited number of weights; often they are available only in *normal* and *bold*. In case, if a font is not available in the specified weight, an alternate will be chosen that is the closest available weight.

Font Variant

The `font-variant` property allows the text to be displayed in a special small-caps variation.

Small-caps or *small capital letters* are slightly different to normal capital letters, in which lowercase letters appear as smaller versions of the corresponding uppercase letters.

Try out the following example to understand how this property actually works:

Example
Try this code »

```
p {
    font-variant: small-caps;
}
```

The value `normal` removes small caps from the text which is already formatted that way.

CSS Text

Formatting Text with CSS

CSS provides several properties that allows you to define various text styles such as color, alignment, spacing, decoration, transformation, etc. very easily and effectively.

The commonly used text properties are: `text-align`, `text-decoration`, `text-transform`, `text-indent`, `line-height`, `letter-spacing`, `word-spacing`, and more.

These properties give you precise control over the visual appearance of the *characters*, *words*, *spaces*, and so on.

Let's see how to set these text properties for an element in more detail.

Text Color

The color of the text is defined by the CSS `color` property.

The style rule in the following example will define the default text color for the page

Example
Try this code »

```
body {
    color: #434343;
}
```

Although, the color property seems like it would be a part of the CSS text, but it is actually a standalone property in CSS. See the tutorial on CSS color to learn more about the color property.

Text Alignment

The `text-align` property is used to set the horizontal alignment of the text.

Text can be aligned in four ways: to the left, right, centre or justified (straight left and right margins).

Let's take a look at an example to understand how this property basically works.

Example
Try this code »

```
h1 {
    text-align: center;
}
p {
    text-align: justify;
}
```

Note: When `text-align` is set to `justify`, each line is stretched so that every line has equal width (except the last line), and the left and right margins are straight. This alignment is generally used in publications such as magazines and newspapers.

Let's take a look at the following illustration to understand what these values actually mean.

Text Decoration

The `text-decoration` property is used to set or remove decorations from text.

This property typically accepts one of the following values: `underline`, `overline`, `line-through`, and `none`. You should avoid underline text that is not a link, as it might confuse the visitor.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
h1 {
    text-decoration: overline;
}
h2 {
    text-decoration: line-through;
}
h3 {
    text-decoration: underline;
```

Removing the Default Underline from Links

The `text-decoration` property is extensively used to remove the default underline from the HTML hyperlinks. You can further provide some other visual cues to make it stand out from the normal text, for example, using dotted border instead of solid underline.

Let's take a look at the following example to understand how it basically works:

Example
Try this code »

```
a {
    text-decoration: none;
    border-bottom: 1px dotted;
}
```

Note: When you create an HTML link, browsers built in style sheet automatically underline it and applies a blue color, so that the readers can clearly see which text is clickable.

Text Transformation

The `text-transform` property is used to set the cases for a text.

Text are often written in mixed case. However, in certain situations you may want to display your text in entirely different case. Using this property you can change an element's text content into uppercase or lowercase letters, or capitalize the first letter of each word without modifying the original text.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
h1 {
```

```
    text-transform: uppercase;
}
h2 {
    text-transform: capitalize;
}
h3 {
    text-transform: lowercase;
}
```

Text Indentation

The `text-indent` property is used to set the indentation of the first line of text within a block of text. It is typically done by inserting the empty space before the first line of text.

The size of the indentation can be specified using percentage (%), length values in pixels, ems, etc.

The following style rule will indent the first line of the paragraphs by 100 pixels.

Example
Try this code »

```
p {
    text-indent: 100px;
}
```

Note: Whether the text is indented *from the left* or *from the right* depends on the direction of text inside the element, defined by the CSS `direction` property.

Letter Spacing

The `letter-spacing` property is used to set extra spacing between the characters of text.

This property can take a length value in pixels, ems, etc. It may also accept negative values. When setting letter spacing, a length value indicates spacing in addition to the default inter-character space.

Let's check out the following example to understand how it really works:

Example
Try this code »

```
h1 {
    letter-spacing: -3px;
}
p {
    letter-spacing: 10px;
}
```

Word Spacing

The `word-spacing` property is used to specify additional spacing between the words.

This property can accept a length value in pixels, ems, etc. Negative values are also allowed.

Let's try out the following example to understand how this property works:

Example
Try this code »

```
p.normal {
    word-spacing: 20px;
}
p.justified {
    word-spacing: 20px;
    text-align: justify;
}
p.preformatted {
    word-spacing: 20px;
    white-space: pre;
}
```

Note: Word spacing can be affected by text justification. Also, even though whitespace is preserved, spaces between words are affected by the `word-spacing` property.

Line Height

The `line-height` property is used to set the height of the text line.

It is also called *leading* and commonly used to set the distance between lines of text.

The value of this property can be a number, a percentage (%), or a length in pixels, ems, etc.

Example
Try this code »

```
p {
    line-height: 1.2;
```

}

When the value is a number, the line height is calculated by multiplying the element's font size by the number. While, percentage values are relative to the element's font size.

Note: Negative values for the `line-height` property are not allowed. This property is also a component of the CSS font shorthand property.

If the value of the `line-height` property is greater than the value of the `font-size` for an element, this difference (called the "*leading*") is cut in half (called the "*half-leading*") and distributed evenly on the top and bottom of the in-line box. Let's check out an example:

Example
Try this code »

```
p {
    font-size: 14px;
    line-height: 20px;
    background-color: #f0e68c;
}
```

See the tutorial on CSS3 text overflow in the advanced section to learn how to handle overflowed text. Also see the CSS3 text shadow section to learn how to apply shadow effect on text.

CSS Links

Styling Links with CSS

Links or hyperlinks are an essential part of a website. It allows visitors to navigate through the site. Therefore styling the links properly is an important aspect of building a user-friendly website.

See the tutorial on HTML links to learn more about links and how to create them.

A link has four different states — `link`, `visited`, `active` and `hover`. These four states of a link can be styled differently through using the following anchor pseudo-class selectors.

- **a:link** — define styles for normal or unvisited links.
- **a:visited** — define styles for links that the user has already visited.
- **a:hover** — define styles for a link when the user place the mouse pointer over it.
- **a:active** — define styles for links when they are being clicked.

You can specify any CSS property you'd like e.g. `color`, `font`, `background`, `border`, etc. to each of these selectors to customize the style of links, just like you do with the normal text.

Example
Try this code »

```
a:link {    /* unvisited link */
    color: #ff0000;
    text-decoration: none;
    border-bottom: 1px solid;
}
a:visited {    /* visited link */
    color: #ff00ff;
```

```
}

a:hover {    /* mouse over link */

    color: #00ff00;

    border-bottom: none;

}

a:active {    /* active link */

    color: #00ffff;

}
```

The order in which you are setting the style for different states of links is important, because what defines last takes precedence over the style rules defined earlier.

Note: In general, the order of the pseudo classes should be the following — `:link`, `:visited`, `:hover`, `:active`, `:focus` in order for these to work properly.

Modifying Standard Link Styles

In all major web browsers such as Chrome, Firefox, Safari, etc. links on the web pages have underlines and uses the browser's default link colors, if you do not set the styles exclusively for them.

By default, text links will appear as follow in most of the browsers:

- An unvisited link as underlined blue text.
- A visited link as underlined purple text.
- An active link as underlined red text.

However, there is no change in the appearance of link in case of the hover state. It remains blue, purple or red depending on which state (i.e. unvisited, visited or active) they are in.

Now let's see how to customize the links by overriding its default styling.

Setting Custom Color of Links

Simply use the CSS `color` property to define the color of your choice for different state of a link. But when choosing colors make sure that user can clearly differentiate between normal text and links.

Let's try out the following example to understand how it basically works:

Example
Try this code »

```
a:link {
    color: #1ebba3;
}
a:visited {
    color: #ff00f4;
}
a:hover {
    color: #a766ff;
}
a:active {
    color: #ff9800;
}
```

Removing the Default Underline from Links

If you don't like the default underline on links, you can simply use the CSS `text-decoration` property to get rid of it. Alternatively, you can apply other styling on links like background color, bottom border, bold font, etc. to make it stand out from the normal text a little better.

The following example shows how to remove or set underline for different states of a link.

Example
Try this code »

```
a:link, a:visited {
    text-decoration: none;
}
a:hover, a:active {
    text-decoration: underline;
}
```

Making Text Links Look Like Buttons

You can also make your ordinary text links look like button using CSS. To do this we need to utilize few more CSS properties such as `background-color`, `border`, `display`, `padding`, etc. You will learn about these properties in detail in upcoming chapters.

Let's check out the following example and see how it really works:

Example
Try this code »

```
a:link, a:visited {
   color: white;
   background-color: #1ebba3;
   display: inline-block;
   padding: 10px 20px;
   border: 2px solid #099983;
   text-decoration: none;
```

```
    text-align: center;
    font: 14px Arial, sans-serif;
}
a:hover, a:active {
    background-color: #9c6ae1;
    border-color: #7443b6Z}
```

CSS Lists

Types of HTML Lists

There are three different types of list in HTML:

- **Unordered lists** — A list of items, where every list items are marked with bullets.
- **Ordered lists** — A list of items, where each list items are marked with numbers.
- **Definition list** — A list of items, with a description of each item.

See the tutorial on HTML lists to learn more about the lists and how to create them.

Styling Lists with CSS

CSS provides the several properties for styling and formatting the most commonly used unordered and ordered lists. These CSS list properties typically allow you to:

- Control the shape or appearance of the marker.
- Specify an image for the marker rather than a bullet point or number.
- Set the distance between a marker and the text in the list.
- Specify whether the marker would appear inside or outside of the box containing the list items.

In the following section we will discuss the properties that can be used to style HTML lists.

Changing the Marker Type of Lists

By default, items in an ordered list are numbered with Arabic numerals (1, 2, 3, 5, and so on), whereas in an unordered list, items are marked with round bullets (•).

But, you can change this default list marker type to any other type such as roman numerals, latin letters, circle, square, and so on using the `list-style-type` property.

Let's try out the following example to understand how this property actually works:

Example
Try this code »

```
ul {
    list-style-type: square;
}
ol {
    list-style-type: upper-roman;
}
```

Changing the Position of List Markers

By default, markers of each list items are positioned `outside` of their display boxes.

However, you can also position these markers or bullet points inside of the list item's display boxes using the `list-style-position` property along with the value `inside`. In this case the lines will wrap under the marker instead of being indented. Here's an example:

Example
Try this code »

```
ol.in li {
    list-style-position: inside;
```

```
}

ol.out li {

    list-style-position: outside;

}
```

Let's take a look at the following illustration to understand how markers or bullets are positioned.

Using Images as List Markers

You can also set an image as a list marker using the `list-style-image` property.

The style rule in the following example assigns a transparent PNG image "arrow.png" as the list marker for all the items in the unordered list. Let's try it out and see how it works:

Example
Try this code »

```
ul li {

    list-style-image: url("images/bullet.png");

}
```

Sometimes, the `list-style-image` property may not produce the expected output. Alternatively, you can use the following solution for better control over the positioning of image markers.

As a work-around, you can also set image bullets via the `background-image` CSS property. First, set the list to have no bullets. Then, define a non-repeating background image for the list element.

The following example displays the image markers equally in all browsers:

Example
Try this code »

```
ul {
    list-style-type: none;
}
ul li {
    background-image: url("images/bullet.png");
    background-position: 0px 5px;    /* X-pos Y-pos (from top-left) */
    background-repeat: no-repeat;
    padding-left: 20px;
}
```

Setting All List Properties At Once

The `list-style` property is a shorthand property for defining all the three properties `list-style-type`, `list-style-image`, and `list-style-position` of a list in one place.

The following style rule sets all the list properties in a single declaration.

Example
Try this code »

```
ul {
    list-style: square inside url("images/bullet.png");
}
```

Note: When using the `list-style` shorthand property, the orders of the values are: `list-style-type` | `list-style-position` | `list-style-image`. It does not matter if one of the values above is missing as long as the rest are in the specified order.

Creating Navigation Menus Using Lists

HTML lists are frequently used to create horizontal navigation bar or menu that typically appear at the top of a website. But since the list items are block elements, so to display them inline we need to use the CSS `display` property. Let's try out an example to see how it really works:

Example
Try this code »

```
ul {
    padding: 0;
    list-style: none;
    background: #f2f2f2;
}
ul li {
    display: inline-block;
}
ul li a {
    display: block;
    padding: 10px 25px;
    color: #333;
    text-decoration: none;
}
ul li a:hover {
    color: #fff;
```

```
    background: #939393;

}
```

CSS Tables

Styling Tables with CSS

Tables are typically used to display tabular data, such as financial reports.

But when you create an HTML table without any styles or attributes, browsers display them without any border. With CSS you can greatly improve the appearance your tables.

CSS provides several properties that allow you to control the layout and presentation of the table elements. In the following section you will see how to use CSS to create elegant and consistent tables.

Adding Borders to Tables

The CSS `border` property is the best way to define the borders for the tables.

The following example will set a black border for the `<table>`, `<th>`, and `<td>` elements.

Example
Try this code »

```
table, th, td {

    border: 1px solid black;

}
```

By default, browser draws a border around the table, as well as around all the cells, with some space in-between, which results in double border. To get rid of this double border problem you can simply collapse the adjoining table cell borders and create clean single line borders.

Let's take a look at the following illustration to understand how a border is applied to a table.

Collapsing Table Borders

There are two distinct models for setting borders on table cells in CSS: *separate* and *collapse*.

In the separate border model, which is the default, each table cell has its own distinct borders, whereas in the collapsed border model, adjacent table cells share a common border. You can set the border model for an HTML table by using the CSS `border-collapse` property.

The following style rules will collapse the table cell borders and apply one pixel black border.

Example
Try this code »

```
table {
   border-collapse: collapse;
}
th, td {
   border: 1px solid black;
}
```

Note: You can also remove the space between the table cell borders through setting the value of CSS `border-spacing` property to 0. However, it only removes the space but do not merge the borders like when you set the `border-collapse` to `collapse`.

Adjusting Space inside Tables

By default, the browser creates the table cells just large enough to contain the data in the cells.

To add more space between the table cell contents and the cell borders, you can simply use the CSS `padding` property. Let's try out the following example and see how it works:

Example
Try this code »

```
th, td {
   padding: 15px;
}
```

You can also adjust the spacing between the borders of the cells using the CSS `border-spacing` property, if the borders of your table are separated (which is default).

The following style rules apply the spacing of 10 pixels between all borders within a table:

Example
Try this code »

```
table {
  border-spacing: 10px;
}
```

Setting Table Width and Height

By default, a table will render just wide and tall enough to contain all of its contents.

However, you can also set the width and height of the table as well as its cells explicitly using the `width` and `height` CSS property. The style rules in the following example will sets the width of the table to 100%, and the height of the table header cells to 40px.

Example
Try this code »

```
table {
  width: 100%;
}
th {
  height: 40px;
}
```

Controlling the Table Layout

A table expands and contracts to accommodate the data contained inside it. This is the default behavior. As data fills inside the table, it continues to expand as long as there is space. Sometimes, however, it is necessary to set a fixed width for the table in order to manage the layout.

You can do this with the help of CSS `table-layout` property. This property defines the algorithm to be used to layout the table cells, rows, and columns. This property takes one of two values:

- **auto** — Uses an automatic table layout algorithm. With this algorithm, the widths of the table and its cells are adjusted to fit the content. This is the default value.
- **fixed** — Uses the fixed table layout algorithm. With this algorithm, the horizontal layout of the table does not depend on the contents of the cells; it only depends on the table's width, the width of the columns, and borders or cell spacing. It is normally faster than auto.

The style rules in the following example specify that the HTML table is laid out using the fixed layout algorithm and has a fixed width of 300 pixels. Let's try it out and see how it works:

Example
Try this code »

```
table {
  width: 300px;
  table-layout: fixed;
}
```

Tip: You can optimize the table rendering performance by specifying the value `fixed` for the `table-layout` property. Fixed value of this property causes the table to be rendered one row at a time, providing users with information at a faster pace.

Note: Without `fixed` value of the `table-layout` property on large tables, users won't see any part of the table until the browser has rendered the whole table.

Aligning the Text Inside Table Cells

You can align text content inside the table cells either horizontally or vertically.

Horizontal Alignment of Cell Contents

For horizontal alignment of text inside the table cells you can use the `text-align` property in the same way as you use with other elements. You align text to either left, right, center or justify.

The following style rules will left-align the text inside the `<th>` elements.

Example
Try this code »

```
th {
  text-align: left;
```

```
}
```

Note: Text inside the `<td>` elements are left-aligned by default, whereas the text inside the `<th>` elements are center-aligned and rendered in bold font by default.

Vertical Alignment of Cell Contents

Similarly, you can vertically align the content inside the `<th>` and `<td>` elements to top, bottom, or middle using the CSS `vertical-align` property. The default vertical alignment is middle.

The following style rules will vertically bottom-align the text inside the `<th>` elements.

Example
Try this code »

```
th {
    height: 40px;
    vertical-align: bottom;
}
```

Controlling the Position of Table Caption

You can set the vertical position of a table caption using the CSS `caption-side` property.

The caption can be placed either at the top or bottom of the table. The default position is top.

Example
Try this code »

```
caption {
    caption-side: bottom;
}
```

Tip: To change the horizontal alignment of the table's caption text (e.g. left or right), you can simply use the CSS `text-align` property, like you do with normal text.

Handling Empty Cells

In tables that uses separate border model, which is default, you can also control the rendering of the cells that have no visible content using the `empty-cells` CSS property.

This property accepts a value of either `show` or `hide`. The default value is `show`, which renders empty cells like normal cells, but if the value `hide` is specified no borders or backgrounds are drawn around the empty cells. Let's try out an example to understand how it really works:

Example
Try this code »

```
table {
    border-collapse: separate;
    empty-cells: hide;
}
```

Note: Placing a non-breaking space (` `) inside a table cell make it non-empty. Therefore, even if that cell looks empty the `hide` value will not hide the borders and backgrounds.

Creating Zebra-striped Tables

Setting different background colors for alternate rows is a popular technique to improve the readability of tables that has large amount of data. This is commonly known as zebra-striping a table.

You can simply achieve this effect by using the CSS `:nth-child()` pseudo-class selector.

The following style rules will highlight every odd rows within the table body.

Example
Try this code »

```
tbody tr:nth-child(odd) {
    background-color: #f2f2f2;
}
```

A zebra-striped table typically looks something like the following picture.

Note: The `:nth-child()` pseudo-class select elements based on their position in a group of siblings. It can take a number, a keyword even or odd, or an expression of the form xn+y where x and y are integers (e.g. 1n, 2n, 2n+1, ...) as an argument.

Making a Table Responsive

Tables are not responsive in nature. However, to support mobile devices you can add responsiveness to your tables by enabling horizontal scrolling on small screens. To do this simply wrap your table with a `<div>` element and apply the style `overflow-x: auto;` as shown below:

Example
Try this code »

```
<div style="overflow-x: auto;">
  <table>
    ... table content ...
  </table>
</div>
```

CSS Box Model

What is Box Model?

Every element that can be displayed on a web page is comprised of one or more rectangular boxes. CSS box model typically describes how these rectangular boxes are laid out on a web page. These boxes can have different properties and can interact with each other in different ways, but every box has a ***content area*** and optional surrounding ***padding***, ***border***, and ***margin areas***.

The following diagram demonstrates how the width, height, padding, border, and margin CSS properties determines how much space an element can take on a web page.

Padding is the transparent space between the element's content and its border (or edge of the box, if it has no border), whereas margin is the transparent space around the border.

Also, if an element has the background color it will be visible through its padding area. The margin area is always remain transparent, it is not affected by the element's background color, however, it causes the background color of the parent element to be seen through it.

Width and Height of the Elements

Usually when you set the width and height of an element using the CSS `width` and `height` properties, in reality you are only setting the width and height of the content area of that element. The actual width and height of the element's box depends on the several factors.

The actual space that an element's box might take on a web page is calculated like this:

Box Size	CSS Properties
Total Width	`width + padding-left + padding-right + border-left + border-right + margin-left + margin-right`
Total Height	`height + padding-top + padding-bottom + border-top + border-bottom + margin-top + margin-bottom`

You will learn about each of these CSS properties in detail in upcoming chapters.

Now let's try out the following example to understand how the box model actually works:

Example
Try this code »

```
div {
    width: 300px;
    height: 200px;
```

```
    padding: 15px; /* set padding for all four sides */

    border: 10px solid black; /* set border for all four sides */

    margin: 20px auto; /* set top and bottom margin to 20 pixels, and left and right margin to auto */

}
```

Note: In CSS box model; the content area of an element's box is the area where its content, e.g., text, image, video, etc. appears. It may also contain descendant elements boxes.

CSS Dimension

Setting Element Dimensions

CSS has several dimension properties, such as `width`, `height`, `max-width`, `min-width`, `max-height`, and `min-height` that allows you to control the width and height of an element. The following sections describe how to use these properties to create a better web page layout.

Setting the Width and Height

The `width` and `height` property defines the width and height of the content area of an element.

This width and height does not include paddings, borders, or margins. See the CSS box model to know how the effective width and height of an element's box is calculated.

Let's try out the following example and see how it actually works:

Example
Try this code »

```
div {
    width: 300px;
    height: 200px;
}
```

The above style rules assigns a fixed width of 300 pixels and height of 200px to the `<div>` element.

The `width` and `height` properties can take the following values:

- *length* - specifies a width in px, em, rem, pt, cm, etc.
- *%* - specifies a width in percentage (%) of the width of the containing element.
- auto - the browser calculates a suitable width for the element.
- initial - Sets the width and height to its default value, which is `auto`.
- inherit - specifies that the width should be inherited from the parent element.

You can not specify negative values to the width and height properties.

Tip: Typically when you create a block element, such as `<div>`, `<p>`, etc. browser automatically set their width to 100% of the available width, and height to whatever is needed to show all the content. You should avoid setting a fixed width and height unless it is necessary.

Setting Maximum Width and Height

You can use the `max-width` and `max-height` property to specify the maximum width and height of the content area. This maximum width and height does not include paddings, borders, or margins.

An element cannot be wider than the `max-width` value, even if the `width` property value is set to something larger. For instance, if the `width` is set to 300px and the `max-width` is set to 200px, the actual width of the element will be 200px. Let's check out an example:

Example
Try this code »

```
div {
    width: 300px;
    max-width: 200px;
}
```

Note: If the `min-width` property is specified with a value greater than that of `max-width` property, in this case the `min-width` value will in fact be the one that's applied.

Likewise, an element that has `max-height` applied will never be taller than the value specified, even if the `height` property is set to something larger. For example, if the `height` is set to 200px and the `max-height` set to 100px, the actual height of the element will be 100px.

Example
Try this code »

```
div {
    height: 200px;
    max-height: 100px;
}
```

Note: If the `min-height` property is specified with a value greater than that of `max-height` property, in this case the `min-height` value will in fact be the one that's applied.

Setting Minimum Width and Height

You can use the `min-width` and `min-height` property specify the minimum width and height of the content area. This minimum width and height does not include paddings, borders, or margins.

An element cannot be narrower than the `min-width` value, even if the `width` property value is set to something lesser. For example, if the `width` is set to 300px and the `min-width` is set to 400px, the actual width of the element will be 400px. Let's see how it actually works:

Example
Try this code »

```
div {
    width: 200px;
    min-width: 300px;
}
```

Note: The `min-width` property is usually used to ensure that an element has at least a minimum width even if no content is present. However the element will be allowed to grow normally if its content exceeds the minimum width set.

Similarly, an element to which `min-height` is applied will never be smaller than the value specified, even if the `height` property is set to something lesser. For example, if the `height` is set to 200px, and the `min-height` is set to 300px, the actual height of the element will be 300px.

Example
Try this code »

Note: The `min-height` property is usually used to ensure that an element has at least a minimum height even if no content is present. However the element will be allowed to grow normally if the content exceeds the minimum height set.

Setting a Width and Height Range

The `min-width` and `min-height` properties are often used in combination with the `max-width` and `max-height` properties to produce a width and height range for an element.

This can be very useful for creating flexible design. In the following example the minimum width of the `<div>` element would be 300px and it can stretches horizontally up to a maximum 500px.

Example
Try this code »

```
div {
    min-width: 300px;
    max-width: 500px;
}
```

Similarly, you can define a height range for an element. In the example below the minimum height of the `<div>` element would be 300px and it can stretches vertically up to a maximum 500px.

Example
Try this code »

```
div {
    min-height: 300px;
    max-height: 500px;
}
```

CSS Padding

CSS Padding Properties

The CSS padding properties allow you to set the spacing between the content of an element and its border (or the edge of the element's box, if it has no defined border).

The padding is affected by the element's `background-color`. For instance, if you set the background color for an element it will be visible through the padding area.

Define Paddings for Individual Sides

You can specify the paddings for the individual sides of an element such as top, right, bottom, and left sides using the CSS `padding-top`, `padding-right`, `padding-bottom`, and the `padding-left` properties, respectively. Let's try out an example to understand how it works:

Example
Try this code »

```
h1 {
    padding-top: 50px;
    padding-bottom: 100px;
}
p {
    padding-left: 75px;
    padding-right: 75px;
}
```

The padding properties can be specified using the following values:

- *length* - specifies a padding in px, em, rem, pt, cm, etc.
- *%* - specifies a padding in percentage (%) of the width of the containing element.
- inherit - specifies that the padding should be inherited from the parent element.

Unlike CSS margin, values for the padding properties cannot be negative.

The Padding Shorthand Property

The `padding` property is a shorthand property to avoid setting padding of each side separately, i.e., `padding-top`, `padding-right`, `padding-bottom` and `padding-left`.

Let's take a look at the following example to understand how it basically works:

Example
Try this code »

```
h1 {
    padding: 50px; /* apply to all four sides */
}
p {
    padding: 25px 75px; /* vertical | horizontal */
}
div {
    padding: 25px 50px 75px; /* top | horizontal | bottom */
}
pre {
    padding: 25px 50px 75px 100px; /* top | right | bottom | left */
}
```

This shorthand notation can take one, two, three, or four whitespace separated values.

- If *one value* is specified, it is applied to **all four sides**.
- If *two values* are specified, the first value is applied to the **top and bottom** side, and the second value is applied to the **right and left** side of the element's box.

- If *three values* are specified, the first value is applied to the **top**, second value is applied to **right and left** side, and the last value is applied to the **bottom**.
- If *four values* are specified, they are applied to the **top**, **right**, **bottom** and the **left** side of the element's box respectively in the specified order.

It is recommended to use the shorthand properties, it will help you to save some time by avoiding the extra typing and make your CSS code easier to follow and maintain.

Effect of Padding and Border on Layout

When creating web page layouts, adding a padding or border to the elements sometimes produce unexpected result, because padding and border is added to the width and height of the box generated by the element, as you have learnt in the CSS box model chapter.

For instance, if you set the width of a `<div>` element to 100% and also apply left right padding or border on it, the horizontal scrollbar will appear. Let's see an example:

Example
Try this code »

```
div {
    width: 100%;
    padding: 25px;
}
```

To prevent padding and border from changing element's box width and height, you can use the CSS `box-sizing` property. In the following example the width and height of the `<div>` box will remain unchanged, however, its content area will decrease with increasing padding or border.

Example
Try this code »

```
div {
```

```
    width: 100%;
    padding: 25px;
    box-sizing: border-box;
}
```

CSS Border

CSS Border Properties

The CSS border properties allow you to define the border area of an element's box.

Borders appear directly between the margin and padding of an element. The border can either be a predefined style like, solid line, dotted line, double line, etc. or an image.

The following section describes how to set the style, color, and width of the border.

Understanding the Different Border Styles

The `border-style` property sets the style of a box's border such as: `solid`, `dotted`, etc. It is a shorthand property for setting the line style for all four sides of the elements border.

The `border-style` property can have the following values: `none`, `hidden`, `solid`, `dashed`, `dotted`, `double`, `inset`, `outset`, `groove`, and `ridge`. Now, let's take a look at the following illustration, it gives you a sense of the differences between the border style types.

The values `none` and `hidden` displays no border, however, there is a slight difference between these two values. In the case of table cell and border collapsing, the `none` value has the *lowest* priority, whereas the `hidden` value has the *highest* priority, if any other conflicting border is set.

The values `inset`, `outset`, `groove`, and `ridge` creates a 3D like effect which essentially depends on the `border-color` value. This is typically achieved by creating a "shadow" from two colors that are slightly lighter and darker than the border color. Let's check out an example:

Example
Try this code »

```
h1 {
    border-style: dotted;
}
p {
    border-style: ridge;
}
```

Note: You must specify a border style in order to make the border appear around an element, because the default border style is `none`. Whereas, the default border width or thickness is `medium`, and the default border color is the same as the text color.

Setting the Border Width

The `border-width` property specifies the width of the border area. It is a shorthand property for setting the thickness of all the four sides of an element's border at the same time.

Let's try out the following example to understand how it works:

Example
Try this code »

```
p {
    border-style: dashed;
    border-width: 10px;
```

```
}
```

Tip: The border width can be specified using any length value, such as px, em, rem, and so on. In addition to length units, the border width may also be specified using one of three keywords: `thin`, `medium`, and `thick`. Percentage values are not allowed.

Specifying the Border Color

The `border-color` property specifies the `color` of the border area. This is also a shorthand property for setting the color of all the four sides of an element's border.

The following style rules adds a solid border of red color around the paragraphs.

Example
Try this code »

```
p {
    border-style: solid;
    border-color: #ff0000;
}
```

Note: The CSS `border-width` or `border-color` property does not work if it is used alone. Use the `border-style` property to set the style of the border first.

The Border Shorthand Property

The `border` CSS property is a shorthand property for setting one or more of the individual border properties `border-width`, `border-style` and `border-color` in a single rule.

Let's take a look at the following example to understand how it works:

Example
Try this code »

```
p {
```

```
    border: 5px solid #00ff00;
}
```

If the value for an individual border property is omitted or not specified while setting the border shorthand property, the default value for that property will be used instead, if any.

For instance, if the value for the `border-color` property is missing or not specified when setting the border, the element's `color` property will be used as the value for the border color.

In the example below, the border will be a solid red line of 5 pixels width:

Example
Try this code »

```
p {
    color: red;
    border: 5px solid;
}
```

But, in the case of `border-style`, omitting the value will cause no border to show at all, because the default value for this property is `none`. In the following example, there will be no border:

Example
Try this code »

```
p {
    border: 5px #00ff00;
}
```

CSS Margin

CSS Margin Properties

The CSS margin properties allow you to set the spacing around the border of an element's box (or the edge of the element's box, if it has no defined border).

An element's margin is not affected by its `background-color`, it is always transparent. However, if the parent element has the background color it will be visible through its margin area.

Setting Margins for Individual Sides

You can specify the margins for the individual sides of an element such as top, right, bottom, and left sides using the CSS `margin-top`, `margin-right`, `margin-bottom`, and the `margin-left` properties, respectively. Let's try out the following example to understand how it works:

Example
Try this code »

```
h1 {
    margin-top: 50px;
    margin-bottom: 100px;
}
p {
    margin-left: 75px;
    margin-right: 75px;
}
```

The margin properties can be specified using the following values:

- *length* - specifies a margin in px, em, rem, pt, cm, etc.
- *%* - specifies a margin in percentage (%) of the width of the containing element.
- auto - the browser calculates a suitable margin to use.
- inherit - specifies that the margin should be inherited from the parent element.

You can also specify negative margins on an element, e.g., `margin: -10px;`, `margin: -5%;`, etc.

The Margin Shorthand Property

The `margin` property is a shorthand property to avoid setting margin of each side separately, i.e., `margin-top`, `margin-right`, `margin-bottom` and `margin-left`.

Let's take a look at the following example to understand how it basically works:

Example
Try this code »

```
h1 {
    margin: 50px; /* apply to all four sides */
}
p {
    margin: 25px 75px; /* vertical | horizontal */
}
div {
    margin: 25px 50px 75px; /* top | horizontal | bottom */
}
hr {
```

```
    margin: 25px 50px 75px 100px; /* top | right | bottom | left */
}
```

This shorthand notation can take one, two, three, or four whitespace separated values.

- If *one value* is specified, it is applied to **all four sides**.
- If *two values* are specified, the first value is applied to the **top and bottom** side, and the second value is applied to the **right and left** side of the element's box.
- If *three values* are specified, the first value is applied to the **top**, second value is applied to **right and left** side, and the last value is applied to the **bottom**.
- If *four values* are specified, they are applied to the **top**, **right**, **bottom** and the **left** side of the element's box respectively in the specified order.

It is recommended to use the shorthand properties, it will help you to save some time by avoiding the extra typing and make your CSS code easier to follow and maintain.

Horizontal Centering with Auto Margins

The `auto` value for the margin property tells the web browser to automatically calculate the margin. This is commonly used to center an element horizontally within a larger container.

Let's try out the following example to understand how it works:

Example
Try this code »

```
div {
    width: 300px;
    background: gray;
    margin: 0 auto;
}
```

The above style rules lets the `<div>` element take up 300 pixels of all the horizontal space available, and the remaining space will be equally divided between left and right margins.

CSS Outline

CSS Outline Properties

The CSS outline properties allow you to define an outline area around an element's box.

An outline is a line that is drawn just outside the border edge of the elements. Outlines are generally used to indicate focus or active states of the elements such as buttons, links, form fields, etc.

The following section describes how to set the style, color, and width of the outline.

Outlines Vs Borders

An outline looks very similar to the border, but it differs from border in the following ways:

- Outlines do not take up space, because they always placed on top of the box of the element which may cause them to overlap other elements on the page.
- Unlike borders, outlines won't allow us to set each edge to a different width, or set different colors and styles for each edge. An outline is the same on all sides.
- Outlines do not have any impact on surrounding elements apart from overlapping.
- Unlike borders, outlines do not change the size or position of the element.
- Outlines may be non-rectangular, but you cannot create circular outlines.

Note: If you put an outline on an element, it will take up the same amount of space on the web pages as if you didn't have an outline on that element. Because it overlap margins (transparent area outside of the border) and surrounding elements.

Understanding the Different Outline Styles

The `outline-style` property sets the style of an element's outline such as: `solid`, `dotted`, etc.

The `outline-style` property can have one of the following values: `none`, `solid`, `dashed`, `dotted`, `double`, `inset`, `outset`, `groove`, and `ridge`. Now, let's take a look at the following illustration, it gives you a sense of the differences between the outline style types.

The value `none` displays no outline. The values `inset`, `outset`, `groove`, and `ridge` creates a 3D like effect which essentially depends on the `outline-color` value. This is typically achieved by creating a "shadow" from two colors that are slightly lighter and darker than the outline color.

Let's try out the following example and see how it basically works:

Example
Try this code »

```
h1 {
    outline-style: dotted;
}
p {
    outline-style: ridge;
}
```

Note: You must specify a outline style in order to make the outline appear around an element, because the default outline style is `none`. Whereas, the default outline width or thickness is `medium`, and the default outline color is the same as the text color.

Setting the Outline Width

The `outline-width` property specifies the width of the outline to be added on an element.

Let's try out the following example to understand how it actually works:

Example
Try this code »

```
p {
    outline-style: dashed;
    outline-width: 10px;
}
```

Tip: The outline width can be specified using any length value, such as px, em, rem, and so on. It can also be specified using one of the three keywords: `thin`, `medium`, and `thick`. Percentage or negative values are not allowed — just like the `border-width` property.

Specifying the Outline Color

The `outline-color` property sets the color of the outline.

This property accepts the same values as those used for the `color` property.

The following style rules adds a solid outline of blue color around the paragraphs.

Example
Try this code »

```
p {
    outline-style: solid;
    outline-color: #0000ff;
}
```

Note: The CSS `outline-width` or `outline-color` property does not work if it is used alone. Use the `outline-style` property to set the style of the outline first.

The Outline Shorthand Property

The `outline` CSS property is a shorthand property for setting one or more of the individual outline properties `outline-style`, `outline-width` and `outline-color` in a single rule.

Let's take a look at the following example to understand how it works:

Example
Try this code »

```
p {
    outline: 5px solid          #ff00ff;
}
```

If the value for an individual outline property is omitted or not specified while setting the outline shorthand property, the default value for that property will be used instead, if any.

For instance, if the value for the `outline-color` property is missing or not specified when setting the outlines, the element's `color` property will be used as the value for the outline color.

In the following example, the outline will be a solid green line of 5px width:

Example
Try this code »

```
p {
    color: green;
    outline: 5px solid;
}
```

But, in the case of `outline-style`, omitting the value will cause no outline to show at all, because the default value for this property is `none`. In the example below, there will be no outline:

Example
Try this code »

```
p {
    outline: 5px #00ff00;
}
```

Removing Outline Around Active Links

The `outline` property is widely used to remove the outline around active links.

However, it is recommended to apply some alternative style to indicate that the link has focus.

Let's try out the following example and see how it basically works:

Example
Try this code »

```
a, a:active, a:focus {
    outline: none;}
```

CSS Cursors

Changing the Look of Cursor

The browsers typically display the mouse pointer over any blank part of a web page, the gloved hand over any linked or clickable item and the edit cursor over any text or text field. With CSS you can redefine those properties to display a variety of different cursors.

Example
Try this code »

```
h1 {
    cursor: move;
}
p {
    cursor: text;
}
```

The table below demonstrates the different cursors that most browsers will accept. Place your mouse pointer over the "TEST" link in the output column to reveal that cursor.

Look	Values	Example	Output
	default	a:hover{cursor:default;}	TEST
	pointer	a:hover{cursor:pointer;}	TEST
	text	a:hover{cursor:text;}	TEST
	wait	a:hover{cursor:wait;}	TEST
	help	a:hover{cursor:help;}	TEST
	progress	a:hover{cursor:progress;}	TEST
	crosshair	a:hover{cursor:crosshair;}	TEST
	move	a:hover{cursor:move;}	TEST

	`url()`	`a:hover{cursor:url("custom.cur"), default;}`	TEST

Creating a Customized Cursor

It is also possible to have completely customized cursors.

The cursor property handles a comma-separated list of user-defined cursors values followed by the *generic cursor*. If the first cursor is specified incorrectly or can't be found, the next cursor in the comma-separated list will be used, and so on until a usable cursor is found.

If none of the user-defined cursors is valid or supported by the browser, the generic cursor at the end of the list will be used instead.

Tip: The standard format that can be used for cursors is the `.cur` format. However, you can convert images such as `.jpg` and `.gif` into `.cur` format using the image converter software freely available online.

Example
Try this code »

```
a {

    cursor: url("custom.gif"), url("custom.cur"), default;

}
```

In the above example `custom.gif` and `custom.cur` is the custom cursor file, uploaded to your server space, and `default` is the generic cursor that will be used if the custom cursor is missing, or isn't supported by the viewer's browser.

Warning: If you are declaring a custom cursor, you must define a *generic cursor* at the end of the list, otherwise the custom cursor will not render correctly.

Here is a live demonstration of a custom cursor.

```
Hover your mouse pointer over this link to reveal the custom cursor
```

Note: IE9 and earlier versions only support URL values of the type `.cur` for static cursor, and `.ani` for animated cursor. However, browsers such as Firefox, Chrome and Safari have support for `.cur`, `.png`, `.gif` and `.jpg` but not `.ani`.

CSS Overflow

The overflow property specifies the behavior that occurs when an element's content overflows (doesn't fit) the element's box.

Handling Overflowing Content

There may be a situation when the content of an element might be larger than the dimensions of the box itself. For example given width and height properties did not allow enough room to accommodate the content of the element.

CSS overflow property allowing you to specify whether to clip content, render scroll bars or display overflow content of a block-level element.

This property can take one of the following values: `visible` (default), `hidden`, `scroll`, and `auto`. CSS3 also defines the `overflow-x` and `overflow-y` properties which allow for independent control of the vertical and horizontal clipping.

Example
Try this code »

```
div {
    width: 250px;
    height: 150px;
    overflow: scroll;
}
```

Value	Description
`visible`	The default value. Content is not clipped; it will be rendered outside the element's box, and may thus overlap other content.
`hidden`	Content that overflows the element's box is clipped and the rest of the content will be invisible.
`scroll`	The overflowing content is clipped, just like hidden, but provides a scrolling mechanism to access the overflowed content.
`auto`	If content overflows the element's box it will automatically provides the scrollbars to see the rest of the content, otherwise scrollbar will not appear.

CSS Units

Understanding CSS Units

The units in which length is measured can be either absolute such as pixels, points and so on, or relative such as percentages (`%`) and `em` units.

Specifying CSS units is obligatory for non-zero values, because there is no default unit. Missing or ignoring a unit would be treated as an error. However, if the value is 0, the unit can be omitted (after all, zero pixels is the same measurement as zero inches).

Note: Lengths refer to distance measurements. A length is a measurement comprising a numeric value and a unit only such as `10px`, `2em`, `50%` etc. The whitespace can't appear between the number and the unit.

Relative Length Units

Relative length units specify a length relative to another length property. Relative units are:

Unit	Description
em	the current font-size
ex	the x-height of the current font

The `em` and `ex` units depend on the font size that's applied to the element.

Using the Em Unit

A measurement of `1em` is equal to the computed value of the `font-size` property of the element on which it is used. It may be used for vertical or horizontal measurement.

For example, if `font-size` of the element set to 16px and `line-height` set to 2.5em then the calculated value of `line-height` in pixel is `2.5 x 16px = 40px`.

Example
Try this code »

```
p {
    font-size: 16px;
    line-height: 2.5em;
}
```

The exception occurs when em is specified in the value of `font-size` property itself, in that case it refers to the font size of the parent element.

So, when we specify a font size in `em`, 1em is equal to the inherited `font-size`. As such, `font-size: 1.2em;` makes the text 1.2 times larger than the parent element's text.

Example
Try this code »

```
body {
```

```
    font-size: 62.5%;

    font-family: Arial, Helvetica, sans-serif;

}

p {

    font-size: 1.6em;

}

p:firt-letter {

    font-size: 3em;

    font-weight: bold;

}
```

Let's understand what this code was all about. The default size for the fonts in all modern browsers is 16px. Our first step is to reduce this size for the entire document by setting the body `font-size` to 62.5%, which resets the font-size to 10px (62.5% of 16px).

This is to round off the default `font-size` for easy `px` to `em` conversion.

Using the Ex Unit

The `ex` unit is equal to the x-height of the current font.

The x-height is so called because it is often equal to the height of the lowercase 'x', as illustrated below. However, an `ex` is defined even for fonts that do not contain an 'x'.

Absolute Length Units

Absolute length units are fixed in relation to each other. They are highly dependent on the output medium, so are mainly useful when the output environment is known. The absolute units consist of the physical units (`in`, `cm`, `mm`, `pt`, `pc`) and the `px` unit.

Unit	Description
`in`	inches – 1in is equal to 2.54cm.
`cm`	centimeters.
`mm`	millimeters.
`pt`	points – In CSS, one point is defined as 1/72 inch (0.353mm).
`pc`	picas – 1pc is equal to 12pt.
`px`	pixel units – 1px is equal to 0.75pt.

Absolute physical units such as `in`, `cm`, `mm`, etc. should be used for print media and similar high-resolution devices. Whereas, for on-screen display such as desktop and lower-resolution devices, it is recommended to use the pixel or em units.

Example

Try this code »

```
h1 { margin: 0.5in; }       /* inches  */

h2 { line-height: 3cm; }    /* centimeters */

h3 { word-spacing: 4mm; }   /* millimeters */

h4 { font-size: 12pt; }     /* points */

h5 { font-size: 1pc; }      /* picas */

h6 { font-size: 12px; }     /* picas */
```

Tip: Style sheets that use the relative units such as *em* or *percentage (%)* can more easily scale from one output environment to another.

CSS Visual Formatting

CSS Visual Formatting Model

The CSS visual formatting model is the algorithm that is used to process the documents for visual media. In the visual formatting model, each element in the document tree generates zero or more boxes according to the box model.

The layout of these boxes is depends on the following factors:

- box dimensions.
- type of the element (block or inline).
- positioning scheme (normal flow, float, and absolute positioning).
- relationships between elements in the document tree.
- external information e.g. viewport size, built-in dimensions of images, etc.

Note: The document tree is the hierarchy of elements encoded in the source document. Every element in the document tree has exactly one parent, with the exception of the root element, which has none.

Type of Boxes Generated in CSS

Every element displayed on a web page generates a rectangular box. The following section describes the types of boxes that may be generated by an element.

Block-level Elements and Block Boxes

Block-level elements are those elements that are formatted visually as blocks (i.e. takes up the full width available), with a line break before and after the element. For example, paragraph element (`<p>`), headings (`<h1> to <h6>`), divisions (`<div>`) etc.

Generally, block-level elements may contain inline elements and other block-level elements.

Inline-level Elements and Inline Boxes

Inline-level elements are those elements of the source document that do not form new blocks of content; the content is distributed in lines. For example, emphasized pieces of text within a paragraph (`<em>`), spans (`<span>`), strong element (`<strong>`) etc.

Inline elements typically may only contain text and other inline elements.

Note: Unlike block-level elements, an inline-level element only takes up as much width as necessary, and does not force line breaks.

CSS Display

CSS Display Property

The CSS specification defines the default display value for all the elements, e.g. the `<div>` element is *rendered as block*, while the `<span>` element is *displayed inline*.

Changing the Default Display Value

Overriding the default display value of an element is an important implication of the `display` property. For example, changing an inline-level element to be displayed as block-level element or changing the block-level element to be displayed as an inline-level element.

Note: The CSS `display` property is one of the most powerful and useful properties in all the CSS. It can be very useful for creating web pages that looks in a different way, but still follow the web standards.

The following section describes you the most commonly used CSS display values.

Display Block

The `block` value of the `display` property forces an element to behave like block-level element, like a `<div>` or `<p>` element. The style rules in the following example displays the `<span>` and `<a>` elements as block-level elements:

Example
Try this code »

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

Note: Changing the display type of an element only changes the display behavior of an element, NOT the type of element it is. For example, an inline element set to `display: block;` is not allowed to have a block element nested inside of it.

Display Inline

The `inline` value of the `display` property causes an element to behave as though it were an inline-level element, like a `<span>` or an `<a>` element. The style rules in the following example displays the `<p>` and `<li>` elements as inline-level elements:

Example
Try this code »

```
p {
    display: inline;
}
```

```
ul li {
    display: inline;
}
```

Display Inline-Block

The `inline-block` value of the `display` property causes an element to generate a block box that will be flowed with surrounding content i.e. in the same line as adjacent content. The following style rules displays the `<div>` and `<span>` elements as inline-block:

Example
Try this code »

```
div {
    display: inline-block;
}
span {
    display: inline-block;
}
```

Display None

The value `none` simply causes an element to generate no boxes at all. Child elements do not generate any boxes either, even if their display property is set to something other than none. The document is rendered as though the element did not exist in the document tree.

Example
Try this code »

```
h1 {
    display: none;
}
p {
    display: none;
}
```

Note: The value `none` for the `display` property does not create an invisible box — it creates no box at all. See the live demo given inside visibility vs display section.

CSS Visibility

Controlling the Visibility of Elements

You can use the `visibility` property to control whether an element is visible or not. This property can take one of the following values listed in the table below:

Value	Description
`visible`	Default value. The box and its contents are visible.
`hidden`	The box and its content are invisible, but still affect the layout of the page.
`collapse`	This value causes the entire row or column to be removed from the display. This value is used for row, row group, column, and column group elements.

`inherit`	Specifies that the value of the visibility property should be inherited from the parent element i.e. takes the same visibility value as specified for its parent.

The style rule `visibility: collapse;` however removes the internal table elements, but it does not affect the layout of the table in any other way. The space normally occupied by the table elements will be filled by the subsequent siblings.

Note: If the style rule `visibility: collapse;` is specified for other elements rather than the table elements, it causes the same behavior as `hidden`.

CSS Visibility vs Display

The display and visibility CSS properties appear to be the same thing, but they are in fact quite different and often confuse those new to web development.

- `visibility: hidden;` hides the element, but it still takes up space in the layout. Child element of a hidden box will be visible if their visibility is set to visible.
- `display: none;` turns off the display and removes the element completely from the document. It does not take up any space, even though the HTML for it is still in the source code. All child elements also have their display turned off, even if their display property is set to something other than none.

Check out the following demo to find out how display and visibility affect the layouts.

CSS Position

CSS Positioning Methods

Positioning elements appropriately on the web pages is a necessity for a good layout design. There are several methods in CSS that you can use for positioning elements. The following section will describe you these positioning methods one by one.

Static Positioning

A static positioned element is always positioned according to the normal flow of the page. HTML elements are positioned static by default. Static positioned elements are not affected by the `top`, `bottom`, `left`, `right`, and `z-index` properties.

Example
Try this code »

```
.box {
    padding: 20px;
    background: #7dc765;
}
```

Relative Positioning

A relative positioned element is positioned relative to its normal position.

In the relative positioning scheme the element's box position is calculated according to the normal flow. Then the box is shifted from this normal position according to the properties — `top` or `bottom` and/or `left` or `right`.

Example
Try this code »

```
.box {
    position: relative;
    left: 100px;
}
```

Note: A relatively positioned element can be moved and overlap other elements, but it keeps the space originally reserved for it in the normal flow.

Absolute Positioning

An absolutely positioned element is positioned relative to the first parent element that has a position other than static. If no such element is found, it will be positioned on a page relative to the 'top-left' corner of the browser window. The box's offsets further can be specified using one or more of the properties `top`, `right`, `bottom`, and `left`.

Absolutely positioned elements are taken out of the normal flow entirely and thus take up no space when placing sibling elements. However, it can overlap other elements depending on the `z-index` property value. Also, an absolutely positioned element can have margins, and they do not collapse with any other margins.

Example
Try this code »

```
.box {
    position: absolute;
    top: 200px;
    left: 100px;
}
```

Fixed Positioning

Fixed positioning is a subcategory of absolute positioning.

The only difference is, a fixed positioned element is fixed with respect to the browser's viewport and does not move when scrolled.

Example
Try this code »

```
.box {
    position: fixed;
    top: 200px;
    left: 100px;
}
```

Note: In case of the print media type, the fixed positioned element is rendered on every page, and is fixed with respect to the page box (even in print-preview). IE7 and IE8 support the fixed value only if a `<!DOCTYPE>` is specified.

CSS Layers

Stacking Elements in Layers Using z-index Property

Usually HTML pages are considered two-dimensional, because text, images and other elements are arranged on the page without overlapping. However, in addition to their horizontal and vertical positions, boxes can be stacked along the z-axis as well i.e. one on top of the other by using the CSS `z-index` property. This property specifies the stack level of a box whose `position` value is one of `absolute`, `fixed`, or `relative`.

The z-axis position of each layer is expressed as an integer representing the stacking order for rendering. An element with a larger `z-index` overlaps an element with a lower one.

A `z-index` property can help you to create more complex webpage layouts. Following is the example which shows how to create layers in CSS.

Example
Try this code »

```
.box {
   position: absolute;
   left: 10px;
   top: 20px;
   z-index: 2;
```

CSS Float

Floating Elements with CSS

You can float elements to the left or right, but only applies to the elements that generate boxes that are not absolutely positioned. Any element that follows the floated element will flow around the floated element on the other side.

The `float` property may have one of the three values:

Value	Description
`left`	The element floats on the left side of its containing block.
`right`	The element floats on the right side of its containing block.
`none`	Removes the float property from an element.

How Elements Float

A floated element is taken out of the normal flow and shifted to the left or right as far as possible in the space available of the containing element.

Other elements normally flow around the floated items, unless they are prevented from doing so by their `clear` property. Elements are floated horizontally, which means that an element can only be floated left or right, not up or down.

Example
Try this code »

```
img {
    float: left;
}
```

If several floating elements are placed adjacently, they will float next to each other if there is horizontal room. If there is not enough room for the float, it is shifted downward until either it fits or there are no more floating elements present.

Example
Try this code »

```
.thumbnail {
    float: left;
    width: 125px;
    height: 125px;
    margin: 10px;
}
```

Turning off Float Using Clear Property

Elements that comes after the floating element will flow around it. The `clear` property specifies which sides of an element's box other floating elements are not allowed.

Example
Try this code »

```
.clear {
    clear: left;
}
```

Note: This property can clear an element only from floated boxes within the same block. It doesn't clear the element from floated child boxes within the element itself. To learn more about clearing float see tutorial on CSS Alignment.

CSS Alignment

Text Alignment

Text inside the block-level elements can aligned by setting the text-align properly.

Example
Try this code »

```
h1 {
    text-align: center;
}
p {
    text-align: left;
}
```

See tutorial on CSS Text to learn more about text formatting.

Center Alignment Using the margin Property

Center alignment of a block-level element is one of the most important implications of the CSS `margin` property. For example, the `<div>` container can be aligned horizontally center by setting the left and right margins to `auto`.

Example
Try this code »

```
div {
    width: 50%;
    margin: 0 auto;
```

}

The style rules in the above example center align the `<div>` element horizontally.

Note: The value `auto` for the `margin` property will not work in Internet Explorer 8 and earlier versions, unless a `<!DOCTYPE>` is specified.

Aligning Elements Using the position Property

The CSS `position` in conjunction with the `left`, `right`, `top` and `bottom` property can be used to align elements with respect to the document's viewport or containing parent element.

Example
Try this code »

```
.up {
    position: absolute;
    top: 0;
}
.down {
    position: absolute;
    bottom: 0;
}
```

To learn more about positioning elements, see the tutorial on CSS positioning.

Left and Right Alignment Using the float Property

The `float` CSS property can be used to align an element to the left or right of its containing block in such a way that other content may flow along its side.

Hence, if an element is floated to the left, content will flow along its right side. Conversely, if the element is floated to the right, content will flow along its left side.

Example
Try this code »

```
div {
    width: 200px;
    float: left;
}
```

Clearing Floats

One of the most confusing things about working with the float-based layouts is the collapsing parent. The parent element doesn't stretch up automatically to accommodate the floated elements. Though, this isn't always obvious if the parent doesn't contain any visually noticeable background or borders, but it is important to be aware of and must dealt with to prevent strange layout and cross-browser problem. See the illustration below:

You can see the `<div>` element doesn't stretch up automatically to accommodate the floated images. This problem can be fixed by clearing the float after the floated elements in the container but before the closing tag of the container element.

Fixing the Collapsed Parent

There are several ways to fix the CSS collapsing parent issue. The following section will describe you these solutions along with the live examples.

Solution 1: Float the Container

The easiest way to fix this problem is to float the containing parent element.

Example
Try this code »

```
.container {
    float: left;
    background: #f2f2f2;
}
```

Warning: This fix will only work in a limited number of circumstances, since floating the parent may produce unexpected results.

Solution 2: Using the Empty Div

This is an old fashioned way but is an easy solution and works across every browser.

Example
Try this code »

```
.clearfix {
    clear: both;
}

/* html code snippet */

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

You could also do this by means of a `<br>` tag. But this method is not recommended since it adds nonsemantic code to your markup.

Solution 3: Using the :after Pseudo-Element

The `:after` pseudo-element in conjunction with the `content` property has been used quite extensively to resolve float-clearing issues.

Example
Try this code »

```
.clearfix:after {
    content: ".";
    display: block;
    height: 0;
    clear: both;
    visibility: hidden;
}
```

The class `.clearfix` is applied to any container that has floating children.

Warning: Internet Explorer up IE7 does not support the `:after` pseudo-element. However IE8 supported, but require a `<!DOCTYPE>` to be declared.

CSS Pseudo-classes

What is Pseudo-class

The CSS pseudo-classes allow you to style the dynamic states of an element such as hover, active and focus state, as well as elements that are existing in the document tree but can't be targeted via the use of other selectors without adding any IDs or classes to them, for example, targeting the first or last child elements.

A pseudo-class starts with a colon (`:`). Its syntax can be given with:

```
selector:pseudo-class { property: value; }
```

The following section describes the most commonly used pseudo-classes.

Anchor Pseudo-classes

Using anchor pseudo-classes links can be displayed in different ways:

These pseudo-classes let you style unvisited links differently from visited ones. The most common styling technique is to remove underlines from visited links.

Example
Try this code »

```
a:link {
    color: blue;
}
a:visited {
    text-decoration: none;
}
```

Some anchor pseudo-classes are dynamic — they're applied as a result of user interaction with the document like on hover, or on focus etc.

Example
Try this code »

```
a:hover {
    color: red;
}
a:active {
    color: gray;
}
```

```
a:focus {
    color: yellow;
}
```

These pseudo-classes change how the links are rendered in response to user actions.

- `:hover` applies when a user places cursor over the element, but does not select it.
- `:active` applies when the element is activated or clicked.
- `:focus` applies when the element has keyboard focus.

Note: To make these pseudo-classes work perfectly, you must define them in the exact order — `:link`, `:visited`, `:hover`, `:active`, `:focus`.

The :first-child Pseudo-class

The `:first-child` pseudo-class matches an element that is the first child element of some other element. The selector `ol li:first-child` in the example below select the first list item of an ordered list and removes the top border form it.

Example
Try this code »

```
ol li:first-child {
    border-top: none;
}
```

Note: To make `:first-child` to work in Internet Explorer 8 and earlier versions, a `<!DOCTYPE>` must be declared at the top of document.

The :last-child Pseudo-class

The `:last-child` pseudo-class matches an element that is the last child element of some other element. The selector `ul li:last-child` in the example below select the last list item from an unordered list and removes the right border from it.

Example
Try this code »

```
ul li:last-child {

    border-right: none;

}
```

Note: The CSS `:last-child` selector does not work in Internet Explorer 8 and earlier versions. Supports in Internet Explorer 9 and above.

The :nth-child Pseudo-class

The CSS3 introduces a new `:nth-child` pseudo-class that allows you to target one or more specific children of a given parent element. The basic syntax of this selector can be given with `:nth-child(N)`, where `N` is an argument, which can be a number, a keyword (`even` or `odd`), or an expression of the form `xn+y` where `x` and `y` are integers (e.g. `1n`, `2n`, `2n+1`, ...).

Example
Try this code »

```
table tr:nth-child(2n) td {

    background: #eee;

}
```

The style rules in the example above simply highlight the alternate table row, without adding any IDs or classes to the `<table>` elements.

Tip: The CSS `:nth-child(N)` selector is very useful in the situations where you have to select the elements that appears inside the document tree in a specific interval or pattern like at even or odd places, etc.

The :lang Pseudo-class

The language pseudo-class `:lang` allows constructing selectors based on the language setting for specific tags. The `:lang` pseudo-class in the example below defines the quotation marks for `<q>` elements that are explicitly given a language value of `no`.

Example
Try this code »

```
q:lang(no) {

    quotes: "~" "~";

}

/* HTML code snippet */

<p>Some text <q lang="no">A quote in a paragraph</q> Some text.</p>
```

Note: Internet Explorer up to version 7 does not support the `:lang` pseudo-class. IE8 supports only if a `<!DOCTYPE>` is specified.

CSS Pseudo-elements

What is Pseudo-element

The CSS pseudo-elements allow you to style the elements or parts of the elements without adding any IDs or classes to them. It will be really helpful in the situations when you just want to style the first letter of a paragraph to create the drop cap effect or you want to insert some content before or after an element, etc. only through style sheet.

CSS3 introduced a new double-colon (`::`) syntax for pseudo-elements to distinguish between them and pseudo-classes. The new syntax of the pseudo-element can be given with:

```
selector::pseudo-element { property: value; }
```

These are the following most commonly used pseudo-elements:

The ::first-line Pseudo-element

The `::first-line` pseudo-element applies special style to the first line of a text.

The style rules in the following example formats the first line of text in a paragraph. The length of first line depends on the size of the browser window or containing element.

Example
Try this code »

```
p::first-line {
    color: #ff0000;
    font-variant: small-caps;
}
```

Note: The CSS properties that can be applied to the `::first-line` pseudo-element are: font properties, background properties, color, word-spacing, letter-spacing, text-decoration, vertical-align, text-transform, line-height.

The ::first-letter Pseudo-element

The `::first-letter` pseudo-element is used to add a special style to the first letter of the first line of a text. The style rules in the following example formats the first letter of the paragraph of text and create the effect like drop cap.

Example
Try this code »

```
p::first-letter {
    color: #ff0000;
    font-size: xx-large;
}
```

Note: The CSS properties that can be applied to the `::first-letter` pseudo-element are: font properties, text-decoration, text-transform, letter-spacing, word-spacing, line-height, float, vertical-align (only if 'float' is 'none'), color, margin and padding properties, border properties, background properties.

The ::before and ::after Pseudo-element

The `::before` and `::after` pseudo-elements can be used to insert generated content either before or after an element's content. The `content` CSS property is used in conjunction with these pseudo-elements, to insert the generated content.

This is very useful for further decorating an element with rich content that should not be part of the page's actual markup. You can insert regular strings of text or an embedded object such as image and other resources using these pseudo-elements.

Example
Try this code »

```
h1::before {
    content: url("images/marker-left.gif");
}

h1::after {
    content: url("images/marker-right.gif");
}
```

CSS Media Types

Introduction to Media Types

One of the most important features of style sheets is that, you can specify separate style sheets for different media types. This is one of the best ways to build printer friendly Web pages — Just assign a different style sheet for the "print" media type.

Some CSS properties are only designed for certain media. For example, the `page-break-after` property only applies to paged media. However there are several properties that may be shared by different media types, but may require different values for that property. The `font-size` property for example can be used for both screen and print media, but possibly with different values.

A document usually needs a larger font on a computer screen as compared to the paper for better readability, also sans-serif fonts are considered easier to read on the screen, while serif fonts are popular for printing. Therefore it is necessary to specify that a style sheet, or a set of style rules, applies to certain media types.

Creating Media Dependent Style Sheets

Three methods are commonly used to specify the media dependencies for style sheets:

Method 1: Using the `@media` At-rules

The `@media` rule is used to define different style rules for different media types in a single style sheet. It is usually followed by a comma-separated list of media types and the CSS declarations block containing the styles rules for target media.

The style declaration in the example below tells the browser to display body content in 14 pixels Arial font on the screen, but in case of printing it will be in a 12 points Times font. However the value of `line-height` property is set to 1.2 for both of them:

Example
Try this code »

```
@media screen{
   body {
      color: #32cd32;
      font-family: Arial, sans-serif;
      font-size: 14px;
   }
}
@media print {
   body {
```

```
        color: #ff6347;
        font-family: Times, serif;
        font-size: 12pt;
    }
}
@media screen, print {
    body {
        line-height: 1.2;
    }
}
```

Note: Style rules outside of `@media` rules apply to all media types that the style sheet applies to. At-rules inside `@media` are invalid in CSS2.1.

Method 2: Using the `@import` At-rules

The `@import` rule is another way of setting style information for a specific target media — Just specify the comma-separated media types after the URL of the imported style sheets.

Example
Try this code »

```
@import url("css/screen.css") screen;
@import url("css/print.css") print;
body {
    background: #f5f5f5;
```

```
    line-height: 1.2;
}
```

The `print` media type in the `@import` statement instructs the browser to load an external style sheet (print.css) and use its styles only for print media.

Note: All `@import` statements must occur at the beginning of a style sheet, before any declarations. Any style rule specified in the style sheet itself override the conflicting style rules in the imported style sheets.

Using the `<link>` element

The `media` attribute on the `<link>` element is used to specify the target media for an external style sheet within the HTML document.

Example
Try this code »

```
<link rel="stylesheet" media="all" href="css/common.css">

<link rel="stylesheet" media="screen" href="css/screen.css">

<link rel="stylesheet" media="print" href="css/print.css">
```

In this example the `media` attribute instructs the browser to load an external style sheet "screen.css" and use its styles only for screen while "print.css" for printing purpose.

Tip: You can also specify multiple media types (each separated with comma e.g. `media="screen, print"`) as well as media quires to the `<link>` element.

Different Media Types

The following table lists the various media types that may used to target different types of devices such as printers, handheld devices, computer screens etc.

Media Type	Description
`all`	Used for all media type devices.
`aural`	Used for speech and sound synthesizers.
`braille`	Used for braille tactile feedback devices.
`embossed`	Used for paged braille printers.
`handheld`	Used for small or handheld devices — usually small screen devices such as mobile phones or PDAs.
`print`	Used for printers.
`projection`	Used for projected presentations, for example projectors.
`screen`	Used primarily for color computer screens.
`tty`	Used for media using a fixed-pitch character grid — such as teletypes, terminals, or portable devices with limited display capabilities.
`tv`	Used for television-type devices — low resolution, color, limited-scrollability screens, sound available.

Warning: However there are several media types to choose from but most of the browser only support `all`, `screen` and `print` media types.

CSS Sprites

What is a Sprite

Sprites are two-dimensional images which are made up of combining small images into one larger image at defined X and Y coordinates.

To display a single image from the combined image, you could use the CSS `background-position` property, defining the exact position of the image to be displayed.

Advantage of Using CSS Image Sprite

A web page with many images, particularly many small images, such as icons, buttons, etc. can take a long time to load and generates multiple server requests.

Using the image sprites instead of separate images will significantly reduce the number of HTTP requests a browser makes to the server, which can be very effective for improving the loading time of web pages and overall site performance.

Note: Reducing the number of HTTP requests has the major impact on reducing response time that makes the web page more responsive for the user.

Checkout the following examples and you will see one visible difference; when you place the mouse pointer over the browser icons in non-sprite version for the first time the hover image will appear after some time, it happens because the hover image is loaded from the server on mouse hover, since the normal and hover images are two different images.

Whereas in sprite version, since all images are combined in a single image the hover image is displayed immediately on mouse hover that results in smooth hover effect.

- Firefox

Chrome

Explorer

Safari

Opera

Firefox
Chrome
Explorer
Safari
Opera

Using CSS sprite technique demonstrated in the: [EXAMPLE - B]; we were able to reduce the number of HTTP requests by 9 and the total file size of the image(s) by 38.2 KB as compared to the [EXAMPLE - A];. That's a pretty huge improvement for such a small example. Imagine what you could do on a complete website.

The whole process of creating this example is explained below.

Making the Image Sprite

We made this sprite image by combining 10 separate images in a single image (mySprite.png). You can create your own sprite using any image editing tool you like.

Tip: For the sake of simplicity, we have used all icons of same size, and place them closely to each other for easy offset calculation.

Display an Icon from Image Sprite

Finally, utilizing CSS, we can display just the part of an image sprite we need.

First of all, we will create the class `.sprite` that will load our sprite image. This is to avoid repetition, because all items share the same background-image.

Example
Try this code »

```
.sprite {

    background: url("images/mySprite.png") no-repeat;

}
```

Now, we must define a class for each item we want to display. For example, to display Internet Explorer icon form the image sprite the CSS code would be.

Example
Try this code »

```
.ie {
    width: 50px; /* Icon width */
    height: 50px; /* Icon height */
    display: inline-block; /* Display icon as inline block */
    background-position: 0 -200px; /* Icon background position in sprite */
}
```

Now the question arises, how did we get those pixel values for `background-position`? Let's find out. The first value is the *horizontal position*, and second is the *vertical position* of background. As the upper-left corner of Internet Explorer icon touches the left edge so its horizontal distance from the starting point i.e. top-left corner of the image sprite is **0**, and since it is placed on the *5th position* so its vertical distance from the starting point of image sprite is `4 X 50px = 200px`, because height of each icon is **50px**.

To show the Internet Explorer icon we have to move its upper-left corner to the starting point i.e. top-left corner of image sprite (mySprite.png). Also, since this icon is placed at the vertical distance of **200px**, we will need to shift the whole background-image (mySprite.png) vertically up by **200px**, which requires us to apply the value as a negative number that is **-200px**, because the negative value makes it go vertically up whereas a positive value would move it down. However, it doesn't require a horizontal offset, since there are no pixels before the upper-left corner of the Internet Explorer icon.

Tip: Just play around with the value of `background-position` property in the upcoming examples and you will quickly learn how the offsets work.

Creating a Navigation Menu Using CSS Image Sprite

In the previous section we have learned, how to display an individual icon from an image sprite. This is the easiest way to use image sprites, now we are going one step ahead by building a navigation menu with *rollover effect* as demonstrated in [EXAMPLE - B].

Here we will use the same sprite image (mySprite.png) to create our navigation menu.

Foundation HTML for Navigation

We will start by creating our navigation menu with an HTML unordered list.

Example
Try this code »

```
<ul class="menu">
    <li class="firefox"><a href="#">Firefox</a></li>
    <li class="chrome"><a href="#">Chrome</a></li>
    <li class="ie"><a href="#">Explorer</a></li>
    <li class="opera"><a href="#">Opera</a></li>
    <li class="safari"><a href="#">Safari</a></li>
</ul>
```

Applying the CSS on Navigation

The following sections will describes you how to convert the simple unordered list given in example above to a spite image based navigation using the CSS.

Step 1: Resetting the List Structure

By default, HTML unordered lists are displayed with bullets. We'll need to remove the default bullets by setting the `list-style-type` attribute to `none`.

Example
Try this code »

```
ul.menu {
    list-style-type: none;
}
ul.menu li {
    padding: 5px;
    font-size: 16px;
    font-family: "Trebuchet MS", Arial, sans-serif;
}
```

Step 2: Setting Common Properties for Each Links

In this step we will set all the common CSS properties that all links are going to share. Such as, `color`, `background-image`, `display`, `padding`, etc.

Example
Try this code »

```
ul.menu li a {
    height: 50px;
    line-height: 50px;
    display: inline-block;
```

```
    padding-left: 60px; /* To sift text off the background-image */

    color: #3E789F;

    background: url("images/mySprite.png") no-repeat; /* As all link share the same background-image */

}
```

Step 3: Setting Default State of Each Links

Now, we must define a class for each menu item, because every item in the image sprite has different `background-position`. For example, Firefox icon is placed at the starting point i.e. top-left corner of the image sprite, so there is no need to shift the background-image. Hence the vertical and horizontal position of the background in this case would be 0. Similarly, you can define background-position for other icons within the image sprite.

Example
Try this code »

```
ul.menu li.firefox a {

    background-position: 0 0;

}

ul.menu li.chrome a {

    background-position: 0 -100px;

}

ul.menu li.ie a {

    background-position: 0 -200px;

}

ul.menu li.safari a {
```

```
    background-position: 0 -300px;
}
ul.menu li.opera a {
    background-position: 0 -400px;
}
```

Step 4: Adding Hover States of Links

Adding hover states owns the same principle as adding the above links. Just move their upper-left corner to the starting point (i.e. top-left corner) of the image sprite as we have done above. You can simply calculate the `background-position` using the following formula:

```
Vertical position of hover state = Vertical position of normal state - 50px
```

As rollover images are just below the default state and height of each icon is equal to 50px. The hover state of icons also doesn't require a horizontal offset, since there are no pixels before the upper-left corner of them.

Example
Try this code »

```
ul.menu li.firefox a:hover {
    background-position: 0 -50px;
}
ul.menu li.chrome a:hover {
    background-position: 0 -150px;
}
ul.menu li.ie a:hover {
    background-position: 0 -250px;
```

```
}
ul.menu li.safari a:hover {
    background-position: 0 -350px;
}
ul.menu li.opera a:hover {
    background-position: 0 -450px;
}
```

Done! Here is our final HTML and CSS code after combining the whole process:

Example

Try this code »

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of Sprite Navigation Menu</title>
<style>
    ul.menu {
        list-style-type: none;
    }
    ul.menu li {
        padding: 5px;
```

```
    font-size: 16px;

    font-family: "Trebuchet MS", Arial, sans-serif;

}
```

CSS Opacity

Cross Browser Opacity

Opacity is now a part of the CSS3 specifications, but it was present for a long time. However, older browsers have different ways of controlling the opacity or transparency.

CSS Opacity in Firefox, Safari, Chrome, Opera and IE9

Here is the most up to date syntax for CSS opacity in all current browsers.

Example
Try this code »

```
p {

    opacity: 0.7;

}
```

The above style rule will make the paragraph element 70% opaque (or 30% transparent).

The opacity property takes a value a value from 0.0 to 1.0. A setting of `opacity: 1;` would make the element completely opaque (i.e. 0% transparent), whereas `opacity: 0;` would make the element completely transparent (i.e. 100% transparent).

CSS Opacity in Internet Explorer 8 and lower

Internet Explorer 8 and earlier version supports a Microsoft-only property "alpha filter" to control the transparency of an element.

Example
Try this code »

```
p {
    filter: alpha(opacity=50);
    zoom: 1;  /* Fix for IE7 */
}
```

Note: Alpha filters in IE accept values from `0` to `100`. The value `0` makes the element completely transparent (i.e. 100% transparent), whereas the value `100` makes the element completely opaque (i.e. 0% transparent).

CSS Opacity for All Browser

Combining the both steps above you will get the *opacity for all browsers*.

Example
Try this code »

```
p {
    opacity: 0.5;  /* Opacity for Modern Browsers */
    filter: alpha(opacity=50);  /* Opacity for IE8 and lower */
    zoom: 1;  /* Fix for IE7 */
}
```

Warning: Including *alpha filter* to control transparency in Internet Explorer 8 and lower versions creates invalid code in your style sheet since this is a Microsoft-only property, not a standard CSS property.

CSS Image Opacity

You can also make transparent images using CSS Opacity.

The three images in the illustration below are all from the same source image. The only differences between them are the level of their opacity.

`opacity:1` `opacity:0.5` `opacity:0.25`

Change Image Opacity on Mouse Over

The following example demonstrates a common use of CSS image opacity, where the opacity of images changes when the user moves the mouse pointer over an image.

— Move your mouse pointer over the images to see the effect.

Text in Transparent Box

When using opacity on an element not only the background of the element that will have transparency, but all of its child elements become transparent as well. It is making the text inside the transparent element hard to read if the value of opacity becomes higher.

OPACITY **OPACITY** **OPACITY** **OPACITY**

To prevent this either you can use transparent PNG images, or put the text block outside of the transparent box and push it visually inside using the negative margin or CSS positioning.

Example
Try this code »

```
div {
  float: left;
  opacity: 0.7;
  border: 1px solid #949781;
}
p {
  float: left;
  position: relative;
  margin-left: -400px;
}
```

CSS Transparency Using RGBA

In addition to RGB CSS3 has introduced a new way RGBA to specify a color that includes alpha transparency as part of the color value. RGBA stands for Red Blue Green Alpha.

The RGBA declaration is a very easy way to set transparency for a color.

Example
Try this code »

```
div {
   background: rgba(200, 54, 54, 0.5);
}
p {
```

```
    color: rgba(200, 54, 54, 0.25);
}
```

The first three numbers representing the color in RGB values i.e. red (0-255), green (0-255), blue (0-255) and the fourth representing alpha transparency value between 0 to 1 (0 makes the color fully transparent , whereas the value of 1 makes it fully opaque).

One important characteristic to note about the RGBA transparency is that — the ability to control the opacity of individual color. With RGBA, we can make the text color of an element transparent and leave background intact.

RGBA **RGBA** **RGBA** **RGBA**

— Or leave the text color alone and change only the transparency of background.

RGBA **RGBA** **RGBA** **RGBA**

You can see how easily you can control the opacity of individual colors rather than the entire element using RGBA. However it is always recommended to define a fallback color for the browsers that do not support the RGBA colors.

Note: The RGBA transparency doesn't affect the child elements the way the `opacity` property does. The alpha value of RGBA affects the transparency of individual color rather than the entire element.

Declaring a Fallback Color

All browsers do not support RGBA colors. However, you can provide an alternative such as solid colors or transparent PNG images for the browsers that don't support it.

Example
Try this code »

```
p {
    /* Fallback for web browsers that doesn't support RGBA */
```

```
    background: rgb(0, 0, 0);

    /* RGBa with 0.5 opacity */

    background: rgba(0, 0, 0, 0.5);

}
```

Warning: Internet Explorer 8 and earlier versions do not support the RGBA colors. They use the gradient filter to achieve the effect of RGBA, which is deprecated.

CSS Attribute Selectors

Understanding the Attribute Selectors

The CSS attribute selectors provides an easy and powerful way to apply the styles on HTML elements based on the presence of a particular attribute or attribute value.

You can create an attribute selector by putting the attribute—optionally with a value—in a pair of square brackets. You can also place an element type selector before it.

The following sections describe the most common attribute selectors.

CSS [attribute] Selector

This is the simplest form of an attribute selector that applies the style rules to an element if a given attribute exists. For example, we can style all the elements that have a `title` attribute by using the following style rules:

Example
Try this code »

```
[title] {

    color: blue;

}
```

The selector `[title]` in the above example matches all elements that has a `title` attribute.

You can also restrict this selection to a particular HTML element by placing the attribute selector after an element type selector, like this:

Example
Try this code »

```
abbr[title] {

    color: red;

}
```

The selector `abbr[title]` matches only `<abbr>` elements that has a `title` attribute, so it matches the abbreviation, but not the anchor elements having `title` attribute.

CSS [attribute="value"] Selector

You can use the `=` operator to make an attribute selector matches any element whose attribute value is exactly equal to the given value:

Example
Try this code »

```
input[type="submit"] {

    border: 1px solid green;

}
```

The selector in the above example matches all `<input>` element that has a `type` attribute with a value equal to `submit`.

CSS [attribute~="value"] Selector

You can use the `~=` operator to make an attribute selector matches any element whose attribute value is a list of *space-separated* values (like `class="alert warning"`) , one of which is exactly equal to the specified value:

Example
Try this code »

```
[class~="warning"] {
    color: #fff;
    background: red;
}
```

This selector matches any HTML element with a `class` attribute that contains space-separated values, one of which is `warning`. For example, it matches the elements having the class values `warning`, `alert warning` etc.

CSS [attribute|="value"] Selector

You can use the `|=` operator to make an attribute selector matches any element whose attribute has a *hyphen-separated* list of values beginning with the specified value:

Example
Try this code »

```
[lang|=en] {
    color: #fff;
    background: blue;
}
```

The selector in the above example matches all elements that has an `lang` attribute containing a value start with `en`, whether or not that value is followed by a hyphen and more characters. In other words, it matches the elements with `lang` attribute that has the values `en`, `en-US`, `en-GB`, and so on but not `US-en`, `GB-en`.

CSS [attribute^="value"] Selector

You can use the `^=` operator to make an attribute selector matches any element whose attribute value *starts with* a specified value. It does not have to be a whole word.

Example
Try this code »

```
a[href^="http://"] {

    background: url("external.png") 100% 50% no-repeat;

    padding-right: 15px;

}
```

The selector in the example above will target all external links and add a small icon indicating that they will open in a new tab or window.

CSS [attribute$="value"] Selector

Similarly, you can use the `$=` operator to select all elements whose attribute value *ends with* a specified value. It does not have to be a whole word.

Example
Try this code »

```
a[href$=".pdf"] {

    background: url("pdf.png") 0 50% no-repeat;

    padding-left: 20px;
```

}

The selector in the example above select all `<a>` elements that links to a PDF document and add a small PDF icon to provide hints to the user about the link.

CSS [attribute*="value"] Selector

You can use the `*=` operator to make an attribute selector matches all elements whose attribute value contains a specified value.

Example
Try this code »

```
[class*="warning"] {

    color: #fff;

    background: red;

}
```

This selector in the example above matches all HTML elements with a `class` attribute that values contains `warning`. For example, it matches the elements having class values `warning`, `alert warning`, `alert-warning` or `alert_warning` etc.

Styling Forms with Attribute Selectors

The attribute selectors are particularly useful for styling forms without `class` or `id`:

Example
Try this code »

```
input[type="text"], input[type="password"] {

    width: 150px;

    display: block;

    margin-bottom: 10px;
```

```
    background: yellow;
}
input[type="submit"] {
    padding: 2px 10px;
    border: 1px solid #804040;
    background: #ff8040;
}
```

CSS Validation

Why Validate Your CSS Code

As a beginner, it is very common that you will make mistake in writing your CSS code. Incorrect or non-standard code may cause unexpected results in how your page displayed or functions in a web browser.

The World Wide Web Consortium (W3C) has created a great tool https://jigsaw.w3.org/css-validator/ to automatically check your style sheets, and point out any problems/errors your code might have, such as invalid CSS property missing closing bracket or missing semicolon (;) etc. It is absolutely free.

Validating a Website

Website validation is the process to ensure that the pages of a website conform to the formal guidelines and standards set by the World Wide Web Consortium (W3C).

There are several specific reasons for validating a website, some of them are:

- It helps to create Web pages that are cross-browser, cross-platform compatible. It also likely to be compatible with the future version of Web browsers and Web standards.
- Standards compliant web pages increase the search engines spider visibility and your pages will more likely be appear in search results.

- It will reduce unexpected errors and make your web pages more accessible to the visitor of your website.

Note: Validation is important. It will ensure that your web pages are interpreted in the same way (the way you want it) by the various web browsers, search engines etc. as well as users and visitors of your Web site.

Follow the link given below to validate your CSS document.

W3.org's CSS Validator

What is JavaScript?

JavaScript ("JS" for short) is a full-fledged dynamic programming language that can add interactivity to a website. It was invented by Brendan Eich (co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation).
JavaScript is versatile and beginner-friendly. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

JavaScript itself is relatively compact, yet very flexible. Developers have written a variety of tools on top of the core JavaScript language, unlocking a vast amount of functionality with minimum effort. These include:

- Browser Application Programming Interfaces (APIs) built into web browsers, providing functionality such as dynamically creating HTML and setting CSS styles; collecting and manipulating a video stream from a user's webcam, or generating 3D graphics and audio samples.
- Third-party APIs that allow developers to incorporate functionality in sites from other content providers, such as Twitter or Facebook.
- Third-party frameworks and libraries that you can apply to HTML to accelerate the work of building sites and applications.

It's outside the scope of this article—as a light introduction to JavaScript—to present the details of how the core JavaScript language is different from the tools listed above. You can learn more in MDN's JavaScript learning area, as well as in other parts of MDN.
The section below introduces some aspects of the core language and offers an opportunity to play with a few browser API features too. Have fun!

A *Hello world!* example

JavaScript is one of the most popular modern web technologies! As your JavaScript skills grow, your websites will enter a new dimension of power and creativity.

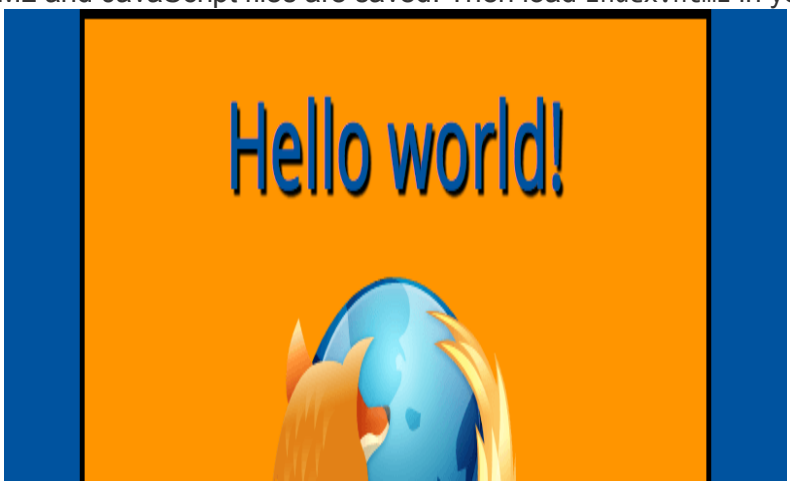
However, getting comfortable with JavaScript is more challenging than getting comfortable with HTML and CSS. You may have to start small, and progress gradually. To begin, let's examine how to add JavaScript to your page for creating a *Hello world!* example. (*Hello world!* is the standard for introductory programming examples.)

Important: If you haven't been following along with the rest of our course, download this example code and use it as a starting point.

1. Go to your test site and create a new folder named `scripts`. Within the scripts folder, create a new file called `main.js`, and save it.
2. In your `index.html` file, enter this code on a new line, just before the closing `</body>` tag:
3.
   ```
   <script src="scripts/main.js"></script>
   ```
4. This is doing the same job as the `<link>` element for CSS. It applies the JavaScript to the page, so it can have an effect on the HTML (along with the CSS, and anything else on the page).
5. Add this code to the `main.js` file:

   ```
   const myHeading = document.querySelector('h1');
   myHeading.textContent = 'Hello world!';
   ```

1. Make sure the HTML and JavaScript files are saved. Then load `index.html` in your browser. You:



Note: The reason the instructions (above) place the `<script>` element near the bottom of the HTML file is that **the browser reads code in the order it appears in the file**.
If the JavaScript loads first and it is supposed to affect the HTML that hasn't loaded yet, there could be problems. Placing JavaScript near the bottom of an HTML page is one way to accommodate this dependency. To learn more about alternative approaches, see Script loading strategies.

What happened?

The heading text changed to *Hello world!* using JavaScript. You did this by using a function called `querySelector()` to grab a reference to your heading, and then store it in a variable called `myHeading`. This is similar to what we did using CSS selectors. When you want to do something to an element, you need to select it first.
Following that, the code set the value of the `myHeading` variable's `textContent` property (which represents the content of the heading) to *Hello world!*.

Note: Both of the features you used in this exercise are parts of the Document Object Model (DOM) API, which has the capability to manipulate documents.

Language basics crash course

To give you a better understanding of how JavaScript works, let's explain some of the core features of the language. It's worth noting that these features are common to all programming languages. If you master these fundamentals, you have a head start on coding in other languages too!

Important: In this article, try entering the example code lines into your JavaScript console to see what happens. For more details on JavaScript consoles, see Discover browser developer tools.

Variables

Variables are containers that store values. You start by declaring a variable with the `var` (less recommended, dive deeper for the explanation) or the `let` keyword, followed by the name you give to the variable:

```
let myVariable;
```

Note: A semicolon at the end of a line indicates where a statement ends. It is only required when you need to separate statements on a single line. However, some people believe it's good practice to have semicolons at the end of each statement. There are other rules for when you should and shouldn't use semicolons. For more details, see Your Guide to Semicolons in JavaScript.
Note: You can name a variable nearly anything, but there are some restrictions. (See this section about naming rules.) If you are unsure, you can check your variable name to see if it's valid.

Note: JavaScript is case sensitive. This means `myVariable` is not the same as `myvariable`. If you have problems in your code, check the case!
Note: For more details about the difference between `var` and `let`, see The difference between var and let.

After declaring a variable, you can give it a value:

```
myVariable = 'Bob';
```

Also, you can do both these operations on the same line:

```
let myVariable = 'Bob';
```

You retrieve the value by calling the variable name:

```
myVariable;
```

After assigning a value to a variable, you can change it later in the code:

```
let myVariable = 'Bob';

myVariable = 'Steve';
```

Note that variables may hold values that have different data types:

Variable	Explanation	Example
String	This is a sequence of text known as a string. To signify that the value is a string, enclose it in single quote marks.	`let myVariable = 'Bob';`
Number	This is a number. Numbers don't have quotes around them.	`let myVariable = 10;`
Boolean	This is a True/False value. The words `true` and `false` are special keywords that don't need quote marks.	`let myVariable = true;`
Array	This is a structure that allows you to store multiple values in a single reference.	`let myVariable = [1,'Bob','Steve',10];` Refer to each member of the array like this: `myVariable[0]`, `myVar`

Variable	Explanation	Example
		`iable[1]`, etc.
Object	This can be anything. Everything in JavaScript is an object and can be stored in a variable. Keep this in mind as you learn.	`let myVariable = document.querySelector('h1');` All of the above examples too.

So why do we need variables? Variables are necessary to do anything interesting in programming. If values couldn't change, then you couldn't do anything dynamic, like personalize a greeting message or change an image displayed in an image gallery.

Comments

Comments are snippets of text that can be added along with code. The browser ignores text marked as comments. You can write comments in JavaScript just as you can in CSS:

/*

Everything in between is a comment.

*/

If your comment contains no line breaks, it's an option to put it behind two slashes like this:

// This is a comment

Operators

An `operator` is a mathematical symbol that produces a result based on two values (or variables). In the following table, you can see some of the simplest operators, along with some examples to try in the JavaScript console.

Operator	Explanation	Symbol(s)	Example
Addition	Add two numbers together or combine two strings.	+	`6 + 9;` `'Hello ' +`

Operator	Explanation	Symbol(s)	Example
			`'world!';`

Subtraction, Multiplication, Division	These do what you'd expect them to do in basic math.	-, *, /	`9 - 3;`
			`8 * 2; // multiply in JS is an asterisk`
			`9 / 3;`

Assignment	As you've seen already: this assigns a value to a variable.	=	`let myVariable = 'Bob';`
Equality	This performs a test to see if two values are equal. It returns a `true`/`false` (Boolean) result.	===	`let myVariable = 3;` `myVariable === 4;`

Not, Does-not-equal	This returns the logically opposite value of what it precedes. It turns a `true` into a `false`, etc.. When it is used alongside the Equality operator, the negation operator tests whether two values are *not* equal.	!, !==	For "Not", the basic expression is `true`, but the comparison returns `false` because we negate it:
			`let myVariable = 3;`
			`!(myVariable === 3);`
			"Does-not-equal" gives basically the same result with different syntax.

Operator	Explanation	Symbol(s)	Example
			Here we are testing "is `myVariable` NOT equal to 3". This returns `false` because `myVariable` IS equal to 3: `let myVariable = 3; myVariable !== 3;`

There are a lot more operators to explore, but this is enough for now. See Expressions and operators for a complete list.

Note: Mixing data types can lead to some strange results when performing calculations. Be careful that you are referring to your variables correctly, and getting the results you expect. For example, enter `'35' + '25'` into your console. Why don't you get the result you expected? Because the quote marks turn the numbers into strings, so you've ended up concatenating strings rather than adding numbers. If you enter `35 + 25` you'll get the total of the two numbers.

Conditionals

Conditionals are code structures used to test if an expression returns true or not. A very common form of conditionals is the `if ... else` statement. For example:

```
let iceCream = 'chocolate';

if(iceCream === 'chocolate') {

  alert('Yay, I love chocolate ice cream!');

} else {

  alert('Awwww, but chocolate is my favorite...');

}
```

The expression inside the `if( ... )` is the test. This uses the identity operator (as described above) to compare the variable `iceCream` with the string `chocolate` to see if the two are equal. If

this comparison returns `true`, the first block of code runs. If the comparison is not true, the second block of code—after the `else` statement—runs instead.

Functions

Functions are a way of packaging functionality that you wish to reuse. It's possible to define a body of code as a function that executes when you call the function name in your code. This is a good alternative to repeatedly writing the same code. You have already seen some uses of functions previously. For example:

```
let myVariable = document.querySelector('h1');

alert('hello!');
```

These functions, `document.querySelector` and `alert`, are built into the browser.
If you see something which looks like a variable name, but it's followed by parentheses— `()` —it is likely a function. Functions often take arguments: bits of data they need to do their job. Arguments go inside the parentheses, separated by commas if there is more than one argument.
For example, the `alert()` function makes a pop-up box appear inside the browser window, but we need to give it a string as an argument to tell the function what message to display.
You can also define your own functions. In the next example, we create a simple function which takes two numbers as arguments and multiplies them:

```
function multiply(num1,num2) {

  let result = num1 * num2;

  return result;

}
```

Try running this in the console; then test with several arguments. For example:

```
multiply(4, 7);

multiply(20, 20);

multiply(0.5, 3);
```

Note: The `return` statement tells the browser to return the `result` variable out of the function so it is available to use. This is necessary because variables defined inside functions are only available inside those functions. This is called variable scoping. (Read more about variable scoping.)

Events

Real interactivity on a website requires event handlers. These are code structures that listen for activity in the browser, and run code in response. The most obvious example is handling the click event, which is fired by the browser when you click on something with your mouse. To demonstrate this, enter the following into your console, then click on the current webpage:

```
document.querySelector('html').onclick = function() {

    alert('Ouch! Stop poking me!');

}
```

There are many ways to attach an event handler to an element. Here we select the `<html>` element, setting its `onclick` handler property equal to an anonymous (i.e. nameless) function, which contains the code we want the click event to run.
Note that

```
document.querySelector('html').onclick = function() {};
```

is equivalent to

```
let myHTML = document.querySelector('html');

myHTML.onclick = function() {};
```

It's just shorter.

Supercharging our example website

With this review of JavaScript basics completed (above), let's add some new features to our example site.

Adding an image changer

In this section, you will learn how to use JavaScript and DOM API features to alternate the display of one of two images. This change will happen as a user clicks the displayed image.

1. Choose an image you want to feature on your example site. Ideally, the image will be the same size as the image you added previously, or as close as possible.

2. Save this image in your `images` folder.
3. Rename the image *firefox2.png*.
4. Add the JavaScript below to your `main.js` file. (Also delete your *Hello world!* JavaScript from the earlier exercise.)

```
let myImage = document.querySelector('img');

myImage.onclick = function() {
    let mySrc = myImage.getAttribute('src');
    if(mySrc === 'images/firefox-icon.png') {
      myImage.setAttribute('src','images/firefox2.png');
    } else {
      myImage.setAttribute('src','images/firefox-icon.png');
    }
}
```

5. Save all files and load `index.html` in the browser. Now when you click the image, it should change to the other one.

This is what happened. You stored a reference to your `<img>` element in the `myImage` variable. Next, you made this variable's `onclick` event handler property equal to a function with no name (an "anonymous" function). So every time this element is clicked:

1. The code retrieves the value of the image's `src` attribute.
2. The code uses a conditional to check if the `src` value is equal to the path of the original image:
1. If it is, the code changes the `src` value to the path of the second image, forcing the other image to be loaded inside the `<img>` element.
2. If it isn't (meaning it must already have changed), the `src` value swaps back to the original image path, to the original state.

Adding a personalized welcome message

Next, let's change the page title to a personalized welcome message when the user first visits the site. This welcome message will persist. Should the user leave the site and return later, we will save the message using the Web Storage API. We will also include an option to change the user, and therefore, the welcome message.

1. In `index.html`, add the following line just before the `<script>` element:

```
<button>Change user</button>
```

2. In `main.js`, place the following code at the bottom of the file, exactly as it is written. This takes references to the new button and the heading, storing each inside variables:

```
let myButton = document.querySelector('button');
let myHeading = document.querySelector('h1');
```

3. Add the function below to set the personalized greeting. This won't do anything yet, but this will change soon.

```
function setUserName() {

  let myName = prompt('Please enter your name.');

  localStorage.setItem('name', myName);
```

```
  myHeading.textContent = 'Mozilla is cool, ' + myName;

}
```

The `setUserName()` function contains a `prompt()` function, which displays a dialog box, similar to `alert()`. This `prompt()` function does more than `alert()`, asking the user to enter data, and storing it in a variable after the user clicks *OK*. In this case, we are asking the user to enter a name. Next, the code calls on an API `localStorage`, which allows us to store data in the browser and retrieve it later. We use localStorage's `setItem()` function to create and store a data item called `'name'`, setting its value to the `myName` variable which contains the user's entry for the name. Finally, we set the `textContent` of the heading to a string, plus the user's newly stored name.

4. Add the `if ... else` block (below). We could call this initialization code, as it structures the app when it first loads.

```
if(!localStorage.getItem('name')) {

  setUserName();

} else {

  let storedName = localStorage.getItem('name');

myHeading.textContent = 'Mozilla is cool, ' + storedName;

}
```

This first line of this block uses the negation operator (logical NOT, represented by the `!`) to check whether the `name` data exists. If not, the `setUserName()` function runs to create it. If it exists (that is, the user set a user name during a previous visit), we retrieve the stored name using `getItem()` and set the `textContent` of the heading to a string, plus the user's name, as we did inside `setUserName()`.

5. Put this `onclick` event handler (below) on the button. When clicked, `setUserName()` runs. This allows the user to enter a different name by pressing the button.

```
myButton.onclick = function() {
  setUserName();
}
```

A user name of null?

When you run the example and get the dialog box that prompts you to enter your user name, try pressing the *Cancel* button. You should end up with a title that reads *Mozilla is cool, null*. This happens because—when you cancel the prompt—the value is set as `null`. *Null* is a special value in JavaScript that refers to the absence of a value.
Also, try clicking *OK* without entering a name. You should end up with a title that reads *Mozilla is cool,* for fairly obvious reasons.

To avoid these problems, you could check that the user hasn't entered a blank name. Update your `setUserName()` function to this:

```
function setUserName() {

  let myName = prompt('Please enter your name.');

  if(!myName) {

    setUserName();

  } else {

    localStorage.setItem('name', myName);

    myHeading.innerHTML = 'Mozilla is cool, ' + myName;

  }

}
```

In human language, this means: If `myName` has no value, run `setUserName()` again from the start. If it does have a value (if the above statement is not true), then store the value in `localStorage` and set it as the heading's text.

Ruby vs. Python: What's the Difference?

Which is better Ruby or Python?

I've used both Ruby and Python in my work — and while they're similar, they're also different in some critical ways. It's a popular question, but an important one, so let me example the difference between Ruby and Python.

Ruby vs. Python What's the Difference?

To set the stage, I first learned web development through Python (and the Python framework called Django). After spending four years building Django apps, I got a job doing Ruby on Rails and expected the transition to be straightforward. That's when it became clear to me that the two languages and frameworks are very different and it's not so easy to jump from one to the other.

So…How are they different?

The Language:

The Ruby on Rails web framework is built using the Ruby programming language while the Django web framework is built using the Python programming language.

This is where many of the differences lay. The two languages are visually similar but are worlds apart in their approaches to solving problems.

Ruby is designed to be infinitely flexible and empowering for programmers. It allows Ruby on Rails to do lots of little tricks to make an elegant web framework. This can feel even magical at times, but the flexibility can also cause some problems. For example, the same magic that makes Ruby work when you don't expect it to can also make it very hard to track down bugs, resulting in hours of combing through code.

Python takes a more direct approach to programming. Its primary goal is to make everything visible to the programmer. This sacrifices some of the elegance that Ruby has but gives Python a big advantage when it comes to learning to code and debugging problems efficiently.

A great example that shows the difference is working with time in your application. Imagine you want to get the time one month from this very second. Here is how you would do that in both languages:

Ruby

```
require 'active_support/all'

new_time = 1.month.from_now
```

Python

```
from datetime import datetime

from dateutil.relativedelta import relativedelta

new_time = datetime.now() + relativedelta(months=1)
```

Notice how Python requires you to import specific functionality from datetime and dateutil libraries. It's explicit, but that's great because you can easily tell where everything is coming from.

With the Ruby version, a lot more is hidden behind a curtain. We import some active_support library and now all of a sudden all integers in Ruby have these ".days" and ".from_now" methods. It reads well, but it's not clear where this functionality came from within active_support. Plus, the idea of patching all integers in the language with new functionality is cool, but it can also cause problems.

Neither approach is right or wrong; they emphasize different things. Ruby showcases the flexibility of the language while Python showcases directness and readability.

Web Frameworks

Django and Rails are both frameworks that help you to build web applications. They have similar performance because both Ruby and Python are scripting languages. Each framework provides you all the concepts from traditional MVC frameworks like models, views, controllers, and database migrations.

Each framework has differences in how you implement these features, but at the core, they are very similar. Python and Ruby also have many libraries you can use to add features to your web applications as well. Ruby has a repository called Rubygems, and Python has a repository called the Package Index.

Community

Python and Ruby have substantial communities behind them. Each community influences the direction of the language, updates, and the way software is built. However, Python has a much broader community than Ruby does. There are a ton of academic use cases in both math and science where Python has thrived, and it continues to grow because of that momentum. Python is also pre-installed on almost every Linux computer making it the perfect language for use on Linux servers (aka. The most popular servers in the world).

Ruby's popularity kicked off when Rails came out in 2005. The community proliferated around Rails and has since been incredibly focused on web development. It has also become more diverse, but not near the level of diversity that Python has reached.

Usage

Who is using these programming languages? Quite a lot of companies. Both Ruby and Python are widespread in the tech world.

There are many famous websites built with Python including Google, Pinterest, Instagram, National Geographic, Mozilla Firefox, and the Washington Post. Similarly, there are just as many Ruby on Rails website examples. Notable companies using Ruby on Rails including Apple, Twitter, Airbnb, Shopify, Github, and Groupon.

Should I learn Python or Ruby first?

Ruby saw a spike in popularity between 2010-2016, but it seems like the industry is trending towards Python. Here's one way to help you make a decision: If you already have a specific client, job, or project lined up that requires you to know Ruby, learn Ruby. If not, learn Python first. Keep in mind there is a difference between Python 2 and Python 3. If you're new to coding then I'd recommend you start with the latest version — Python 3

Conclusion: Ruby vs. Python?

Anything you can do in Ruby on Rails you could also do in Python and Django. Which framework is better isn't a question of capability. The better question might be: which language is better suited for your or your team?

If you plan on sticking with building web applications, then consider prioritizing Ruby on Rails. The community is good and they are always on the bleeding edge. If you are interested in building web

applications but would like to learn a language that's more widely applicable and proficient with handing data analytics than I'd recommend you learn Python.

	RUBY	PYTHON
LANGUAGE	• More magical • Created in 1995 by Yukihiro Matsumoto	• More Direct • Created in 1991 by Guido Van Rossum
PROS	• Tons of features out of the box for web development • Quick to embrace new things	• Very easy to learn • A diverse community with big ties to Linux and academia
CONS	• Can be very hard to debug at times	• Often very explicit and inelegant to read
WEB FRAMEWORKS	• Ruby on Rails-Started in 2005 by David Heinemeier Hansson	• Django-Started in 2003 by Adrian Holovaty and Simon Willison
COMMUNITY	• Innovates quicker but causes more things to break • Very web focused	• Very stable and diverse but innovates slower • Used widely in academia and Linux
USAGE	• Apple • Twitter • Github • Airbnb • Github • Groupon • Shopify	• Google • Pinterest • YouTube • Dropbox • National Geographic • The Washington Post

CONCLUSION

If you have followed all the instructions in this book, you should end up with a page that looks something like the image below. You can als