Introducción a CSS

Curso de Fundamentos

Introducción

En la lección anterior, aprendiste a escribir el código HTML que determina cómo se estructura una página web. El siguiente paso es hacer que esa estructura se vea bien con algo de *estilo* , que es exactamente para lo que sirve CSS. En las próximas lecciones, nos centraremos en lo que creemos que son algunos conceptos básicos de CSS, cosas que todos deberían saber desde el principio, ya sea que estén comenzando o necesiten un repaso.

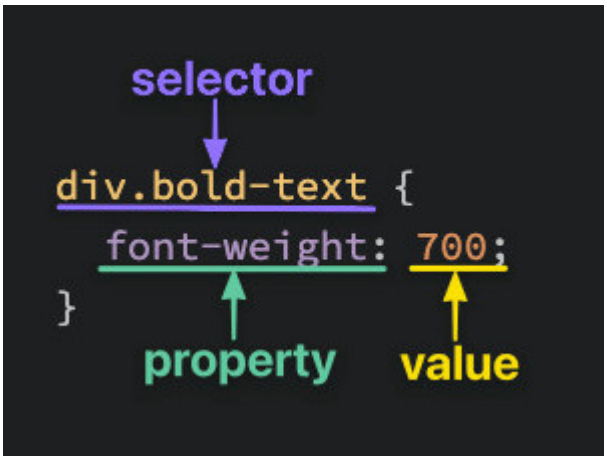
Resumen de la lección

Esta sección contiene una descripción general de los temas que aprenderá en esta lección.

- Añadir estilos a HTML con CSS.
- Comprenda cómo utilizar los atributos de clase e ID.
- Agregue estilos a elementos específicos utilizando los selectores correctos.

Sintaxis básica

En el nivel más básico, CSS se compone de varias reglas. Estas reglas se componen de un selector (más sobre esto en breve) y una lista de declaraciones separadas por punto y coma, y cada una de esas declaraciones se compone de un par propiedad-valor.



> **Nota**
>
> A `<div>` es uno de los elementos HTML básicos. Es un contenedor vacío. En general, es mejor utilizar otras etiquetas como `<h1>` o `<p>` para el contenido de sus proyectos, pero a medida que aprenda más sobre CSS, descubrirá que hay muchos casos en los que lo que necesita es simplemente un contenedor para otros elementos. Muchos de nuestros ejercicios utilizan `<div>` s simples para simplificar. En lecciones posteriores, profundizaremos mucho más sobre cuándo es apropiado utilizar los distintos elementos HTML.

Selectores

Los selectores hacen referencia a los elementos HTML a los que se aplican las reglas CSS; son lo que realmente se "selecciona" para cada regla. Las siguientes subsecciones no cubren todos los selectores disponibles, pero son, con diferencia, los más comunes y los que deberías acostumbrarte a utilizar primero.

Selector universal

El selector universal seleccionará elementos de cualquier tipo, de ahí el nombre "universal", y su sintaxis es un simple asterisco. En el ejemplo siguiente, se aplicaría el `color: purple;` estilo a cada elemento.

```
* {
  color: purple;
}
```

Selectores de tipo

Un selector de tipo (o selector de elementos) seleccionará todos los elementos del tipo de elemento dado, y la sintaxis es solo el nombre del elemento:

```
<!-- index.html -->

<div>Hello, World!</div>
<div>Hello again!</div>
<p>Hi...</p>
<div>Okay, bye.</div>
```

```
/* styles.css */

div {
  color: white;
}
```

Aquí `<div>` se seleccionarían los tres elementos, mientras que el `<p>` elemento no.

Selectores de clase

Los selectores de clase seleccionarán todos los elementos con la clase dada, que es simplemente un atributo que se coloca en un elemento HTML. Aquí se explica cómo agregar una clase a una etiqueta HTML y seleccionarla en CSS:

```
<!-- index.html -->

<div class="alert-text">Please agree to our terms of service.<
```

```
/* styles.css */

.alert-text {
  color: red;
}
```

Tenga en cuenta la sintaxis de los selectores de clase: un punto seguido inmediatamente del valor del atributo de clase, que distingue entre mayúsculas y minúsculas. No es necesario que las clases sean específicas de un elemento en particular, por lo que puede utilizar la misma clase en todos los elementos que desee.

Another thing you can do with the class attribute is to add multiple classes to a single element as a space-separated list, such as `class="alert-text severe-alert"`. Since whitespace is used to separate class names like this, you should never use spaces for multi-worded names and should use a hyphen instead.

ID selectors

ID selectors are similar to class selectors. They select an element with the given ID, which is another attribute you place on an HTML element. The major difference between classes and IDs is that an element can only have **one** ID. It cannot be repeated on a single page and should not contain any whitespace:

```
<!-- index.html -->

<div id="title">My Awesome 90's Page</div>
```

```
/* styles.css */

#title {
  background-color: red;
}
```

For IDs, instead of a period, we use a hashtag immediately followed by the case-sensitive value of the ID attribute. A common pitfall is people overusing the ID attribute when they don't necessarily need to, and when classes will suffice. While there are cases where using an ID makes sense or is needed, such as taking advantage of specificity or having links redirect to a section on the current page, you should use IDs **sparingly** (if at all).

The grouping selector

What if we have two groups of elements that share some of their style declarations?

```
.read {
  color: white;
  background-color: black;
  /* several unique declarations */
}

.unread {
  color: white;
  background-color: black;
  /* several unique declarations */
}
```

Both our `.read` and `.unread` selectors share the `color: white;` and `background-color: black;` declarations, but otherwise have several of their own unique declarations. To cut down on the repetition, we can group these two selectors together as a comma-separated list:

```
.read,
.unread {
  color: white;
  background-color: black;
}

.read {
  /* several unique declarations */
}

.unread {
  /* several unique declarations */
}
```

Both of the examples above (with and without grouping) will have the same result, but the second example reduces the repetition of declarations and makes it easier to edit either the `color` or `background-color` for both classes at once.

Chaining selectors

Another way to use selectors is to chain them as a list without any separation. Let's say we had the following HTML:

```
<div>
  <div class="subsection header">Latest Posts</div>
  <p class="subsection preview">This is where a preview for a
</div>
```

We have two elements with the `subsection` class that have some sort of unique styles, but what if we only want to apply a separate rule to the element that also has `header` as a second class? Well, we could chain both the class selectors together in our CSS like so:

```
.subsection.header {
  color: red;
}
```

What `.subsection.header` does is it selects any element that has both the `subsection` *and* `header` classes. Notice how there isn't any space between the `.subsection` and `.header` class selectors. This syntax basically works for chaining any combination of selectors, except for chaining more than one type selector.

This can also be used to chain a class and an ID, as shown below:

```
<div>
  <div class="subsection header">Latest Posts</div>
  <p class="subsection" id="preview">
    This is where a preview for a post might go.
  </p>
</div>
```

You can take the two elements above and combine them with the following:

```css
.subsection.header {
  color: red;
}

.subsection#preview {
  color: blue;
}
```

In general, you can't chain more than one type selector since an element can't be two different types at once. For example, chaining two type selectors like `div` and `p` would give us the selector `divp`, which wouldn't work since the selector would try to find a literal `<divp>` element, which doesn't exist.

Descendant combinator

Combinators allow us to combine multiple selectors differently than either grouping or chaining them, as they show a relationship between the selectors. There are four types of combinators in total, but for right now we're going to only show you the **descendant combinator**, which is represented in CSS by a single space between selectors. A descendant combinator will only cause elements that match the last selector to be selected if they also have an ancestor (parent, grandparent, etc.) that matches the previous selector.

So something like `.ancestor .child` would select an element with the class `child` if it has an ancestor with the class `ancestor`. Another way to think of it is that `child` will only be selected if it is nested inside `ancestor`, regardless of how deep that nesting is. Take a quick look at the example below and see if you can tell which elements would be selected based on the CSS rule provided:

```html
<!-- index.html -->

<div class="ancestor">
  <!-- A -->
  <div class="contents">
    <!-- B -->
    <div class="contents"><!-- C --></div>
  </div>
```

```
</div>

<div class="contents"><!-- D --></div>
```

```
/* styles.css */

.ancestor .contents {
  /* some declarations */
}
```

In the above example, the first two elements with the `contents` class (B and C) would be selected, but that last element (D) wouldn't be. Was your guess correct?

There's really no limit to how many combinators you can add to a rule, so `.one .two .three .four` would be totally valid. This would just select an element that has a class of `four` if it has an ancestor with a class of `three`, and if that ancestor has its own ancestor with a class of `two`, and so on. You generally want to avoid trying to select elements that need this level of nesting, though, as it can get pretty confusing and long, and it can cause issues when it comes to specificity.

Properties to get started with

There are some CSS properties that you're going to be using all the time, or at the very least more often than not. We're going to introduce you to several of these properties, though this is by no means a complete list. Learning the following properties will be enough to help get you started.

Color and background-color

The `color` property sets an element's text color, while `background-color` sets, well, the background color of an element. I guess we're done here?

Almost. Both of these properties can accept one of several kinds of values. A common one is a keyword, such as an actual color name like `red` or the `transparent` keyword. They also accept HEX, RGB, and HSL values, which you may be familiar with if you've ever used a photoshop program or a site where you could customize your profile colors.

```css
p {
  /* hex example: */
  color: #1100ff;
}

p {
  /* rgb example: */
  color: rgb(100, 0, 127);
}

p {
  /* hsl example: */
  color: hsl(15, 82%, 56%);
}
```

Take a quick look at CSS Legal Color Values to see how you can adjust the opacity of these colors by adding an alpha value.

Typography basics and text-align

`font-family` can be a single value or a comma-separated list of values that determine what font an element uses. Each font will fall into one of two categories, either a "font family name" like `"Times New Roman"` (we use quotes due to the whitespace between words) or a "generic family name" like `serif` (generic family names never use quotes).

If a browser cannot find or does not support the first font in a list, it will use the next one, then the next one and so on until it finds a supported and valid font. This is why it's best practice to include a list of values for this property, starting with the font you want to be used most and ending with a generic font family as a fallback, e.g. `font-family: "Times New Roman", serif;`

`font-size` will, as the property name suggests, set the size of the font. When giving a value to this property, the value should not contain any whitespace, e.g. `font-size: 22px` has no space between "22" and "px".

`font-weight` affects the boldness of text, assuming the font supports the specified weight. This value can be a keyword, e.g. `font-weight: bold` , or a number between 1

and 1000, e.g. `font-weight: 700` (the equivalent of `bold`). Usually, the numeric values will be in increments of 100 up to 900, though this will depend on the font.

`text-align` will align text horizontally within an element, and you can use the common keywords you may have come across in word processors as the value for this property, e.g. `text-align: center`.

Image height and width

Images aren't the only elements that we can adjust the height and width on, but we want to focus on them specifically in this case.

By default, an `<img>` element's `height` and `width` values will be the same as the actual image file's height and width. If you wanted to adjust the size of the image without causing it to lose its proportions, you would use a value of "auto" for the `height` property and adjust the `width` value:

```
img {
  height: auto;
  width: 500px;
}
```

For example, if an image's original size was 500px height and 1000px width, using the above CSS would result in a height of 250px.

These properties work in conjunction with the height and width attributes in the HTML. It's best to include both of these properties and the HTML attributes for image elements, even if you don't plan on adjusting the values from the image file's original ones. When these values aren't included, if an image takes longer to load than the rest of the page contents, it won't take up any space on the page at first but will suddenly cause a drastic shift of the other page contents once it does load in. Explicitly stating a `height` and `width` prevents this from happening, as space will be "reserved" on the page and appear blank until the image loads.

Adding CSS to HTML

Now that we've learned some basic syntax, you might be wondering *how* to add all this CSS to our HTML. There are three methods to do so.

External CSS

External CSS is the most common method you will come across, and it involves creating a separate file for the CSS and linking it inside of an HTML's opening and closing `<head>` tags with a void `<link>` element:

```
<!-- index.html -->

<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

```
/* styles.css */

div {
  color: white;
  background-color: black;
}

p {
  color: red;
}
```

First, we add a void `<link>` element inside of the opening and closing `<head>` tags of the HTML file. The `href` attribute is the location of the CSS file, either an absolute URL or, what you'll be utilizing, a URL relative to the location of the HTML file. In our example above, we are assuming both files are located in the same directory. The `rel` attribute is required, and it specifies the relationship between the HTML file and the linked file.

Then inside of the newly created `styles.css` file, we have the selector (the `div` and `p`), followed by a pair of opening and closing curly braces, which create a "declaration block". Finally, we place any declarations inside of the declaration block. `color: white;` is one declaration, with `color` being the property and `white` being the value, and `background-color: black;` is another declaration.

A note on file names: `styles.css` is just what we went with as the file name here. You can name the file whatever you want as long as the file type is `.css`, though "style" or "styles" is most commonly used.

A couple of the pros to this method are:

1. It keeps our HTML and CSS separated, which results in the HTML file being smaller and making things look cleaner.
2. We only need to edit the CSS in *one* place, which is especially handy for websites with many pages that all share similar styles.

Internal CSS

Internal CSS (or embedded CSS) involves adding the CSS within the HTML file itself instead of creating a completely separate file. With the internal method, you place all the rules inside of a pair of opening and closing `<style>` tags, which are then placed inside of the opening and closing `<head>` tags of your HTML file. Since the styles are being placed directly inside of the `<head>` tags, we no longer need a `<link>` element that the external method requires.

Besides these differences, the syntax is exactly the same as the external method (selector, curly braces, declarations):

```
<head>
  <style>
    div {
      color: white;
      background-color: black;
    }

    p {
      color: red;
    }
  </style>
</head>
<body>
  ...
</body>
```

This method can be useful for adding unique styles to a *single page* of a website, but it doesn't keep things separate like the external method, and depending on how many rules and declarations there are it can cause the HTML file to get pretty big.

Inline CSS

Inline CSS makes it possible to add styles directly to HTML elements, though this method isn't as recommended:

```
<body>
  <div style="color: white; background-color: black;">...</div
</body>
```

The first thing to note is that we don't actually use any selectors here, since the styles are being added directly to the opening `<div>` tag itself. Next, we have the `style=` attribute, with its value within the pair of quotation marks being the declarations.

Si necesita agregar un estilo *único a un solo* elemento, este método puede funcionar bien. Sin embargo, en general, no es exactamente una forma recomendada de agregar CSS a HTML por algunas razones:

- Puede volverse bastante complicado rápidamente una vez que comienzas a agregar muchas *declaraciones* a un solo elemento, lo que provoca que tu archivo HTML se hinche innecesariamente.
- Si desea que muchos elementos tengan el mismo estilo, tendrá que copiar y pegar el mismo estilo en cada elemento individual, lo que provocará muchas repeticiones innecesarias y más hinchazón.
- Cualquier CSS en línea anulará los otros dos métodos, lo que puede generar resultados inesperados. (Si bien no profundizaremos en esto aquí, esto puede aprovecharse).

Asignación

1. Vaya a nuestro repositorio de ejercicios CSS y lea el archivo README.

2. Luego, una vez que sepa cómo utilizar los ejercicios, navegue hasta el directorio del repositorio de ejercicios CSS `foundations/intro-to-css` . Revise cada archivo README antes de completar los siguientes ejercicios en orden:

   - `01-css-methods`
   - `02-class-id-selectors`
   - `03-group-selectors`
   - `04-chain-selectors`
   - `05-descendant-combinator`

   Nota: Las soluciones de estos ejercicios se pueden encontrar en la `solution` carpeta de cada ejercicio.

Comprobación de conocimientos

Las siguientes preguntas son una oportunidad para reflexionar sobre temas clave de esta lección. Si no puede responder una pregunta, haga clic en ella para revisar el material, pero tenga en cuenta que no se espera que memorice o domine este conocimiento.

- ¿Cuál es la sintaxis de los selectores de clase e ID?
- ¿Cómo aplicarías una sola regla a dos selectores diferentes?
- Dado un elemento que tiene un id de `title` y una clase de `primary` , ¿cómo utilizarías ambos atributos para una sola regla?
- ¿Qué hace el combinador descendiente?
- ¿Cuáles son los nombres de las tres formas de agregar CSS a HTML?
- ¿Cuáles son las principales diferencias entre las tres formas de agregar CSS a HTML?

Recursos adicionales

Esta sección contiene enlaces útiles a contenido relacionado. No es obligatoria, por lo que se la puede considerar complementaria.

- Los valores y unidades de Mozilla CSS se pueden utilizar para aprender los distintos tipos de valores posibles en términos absolutos o relativos.
- Un Scrim interactivo que cubre gran parte del material de la lección en forma interactiva.

Mejorar en GitHub Informar un problema Ver registro de cambios de la lección

Ver curso

Marcar como completado

Próxima lección

¡Apóyanos!

El Proyecto Odin está financiado por la comunidad. ¡Únase a nosotros para ayudar a estudiantes de todo el mundo apoyando el Proyecto Odin!

Más información Dona ahora

THE ODIN PROJECT

Educación en codificación de alta calidad mantenida por una comunidad de código abierto.

Sobre nosotros

Acerca de

Equipo

Blog

Casos de éxito

Apoyo

Preguntas frecuentes

Contribuir

Contáctenos

Guías

Guías de la comunidad

Guías de instalación

Legal

Términos

Privacidad