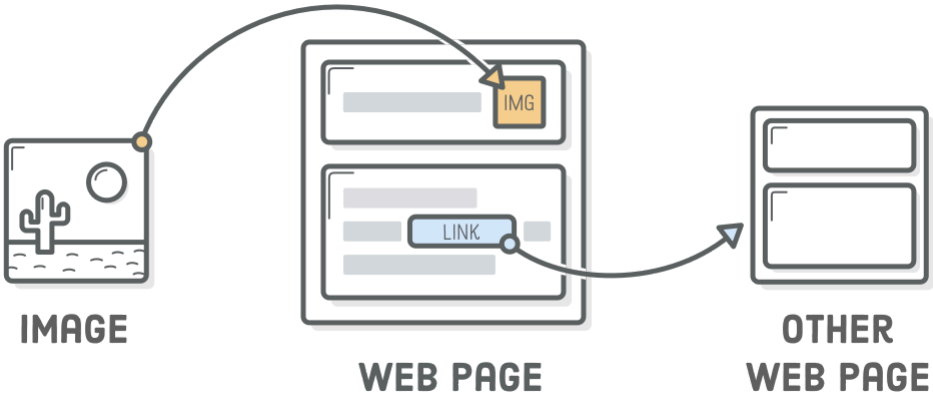
ENLACES E IMÁGENES

Nº 3. EL HTML Y EL CSS SON DIFÍCILES

Un tutorial HTML fácil de usar sobre cómo conectar páginas web entre sí

En el capítulo anterior se trataron algunos elementos HTML muy importantes, pero sólo estábamos tratando con una única página web. Los enlaces y las imágenes son fundamentalmente diferentes de esos elementos en el sentido de que tratan con recursos *externos* . Los enlaces dirigen al usuario a un documento HTML diferente y las imágenes atraen otro recurso a la página.

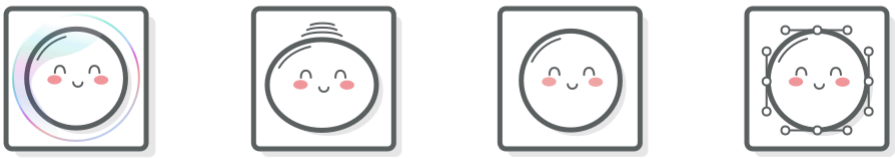


Para utilizar enlaces e imágenes, también necesitaremos conocer otro componente de la sintaxis HTML: los atributos. Los atributos abrirán un nuevo mundo de posibilidades para nuestras páginas web.

En este capítulo, crearemos un sitio web simple compuesto por varios documentos HTML y archivos de imagen. Puede que se llame *Enlaces e imágenes* , pero el tema central de este capítulo es en realidad la organización de archivos y carpetas. A medida que empecemos a trabajar con varios archivos, descubriremos la importancia de ser un desarrollador web organizado.

CONFIGURACIÓN

Este capítulo trata sobre cómo vincular páginas web, por lo que necesitaremos crear algunos archivos HTML nuevos antes de codificar nada. Trabajaremos con tres páginas web independientes en este capítulo, junto con algunos archivos de imágenes de varios formatos:



Para comenzar, crea una nueva carpeta llamada `links-and-images`para almacenar todos nuestros archivos. Deberías poder hacer esto en Atom usando el mismo proceso que seguimos en el capítulo Introducción .

PÁGINA DE ENLACES

A continuación, agregue un nuevo archivo a esa carpeta `links.htmly` coloque la siguiente plantilla HTML. Esta plantilla le resultará familiar por el capítulo anterior.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Links</title>
  </head>
  <body>
    <h1>Links</h1>
  </body>
</html>
```

PÁGINA DE IMÁGENES

En la misma carpeta, crea otro archivo llamado `images.html`:

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

```
</html>
```

PÁGINA DE EXTRAS

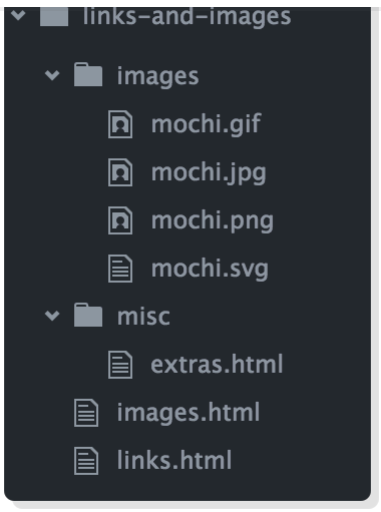
Nuestra última página nos ayudará a demostrar los vínculos relativos. Cree una nueva carpeta en `links-and-images`llamada `misc`, luego agregue un nuevo archivo llamado `extras.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Extras</title>
  </head>
  <body>
    <h1>Extras</h1>
  </body>
</html>
```

Tenga en cuenta que puede crear una nueva carpeta en Atom haciendo clic derecho en el panel del explorador de archivos y seleccionando **Nueva carpeta** en el menú contextual. La vida es mejor cuando nunca necesita salir de su editor de texto.

DESCARGAS DE IMÁGENES

Incorporaremos imágenes en nuestro `images.html`archivo, así que asegúrate de descargar también estas imágenes de mochi de ejemplo . Descomprímelas en tu `links-and-images`carpeta, conservando la `images`carpeta principal del archivo ZIP. Tu proyecto ahora debería verse así:



ANCLAS

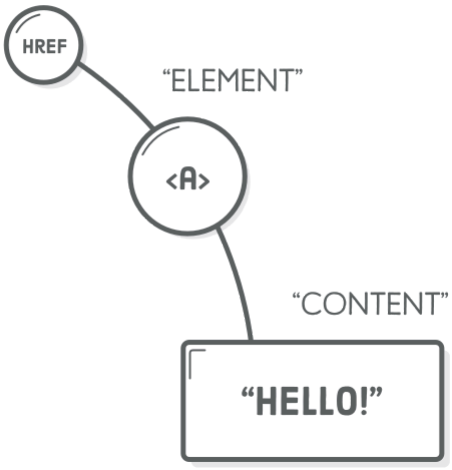
Los enlaces se crean con el `<a>`elemento , que significa "ancla". Funciona igual que todos los elementos del capítulo anterior: cuando envuelves un texto entre `<a>`etiquetas, modificas el significado de ese contenido. Echemos un vistazo añadiendo el siguiente párrafo al `<body>`elemento of `links.html`:

```
<p>This example is about links and <a>images</a>.</p>
```

If you load the page in a web browser, you'll notice that the `<a>` element doesn't look like a link at all. Yes, unfortunately, the `<a>` element on its own doesn't do much of anything.

LINKS

In the same way that an element adds meaning to the content it contains, an HTML "attribute" adds meaning to the element it's attached to.

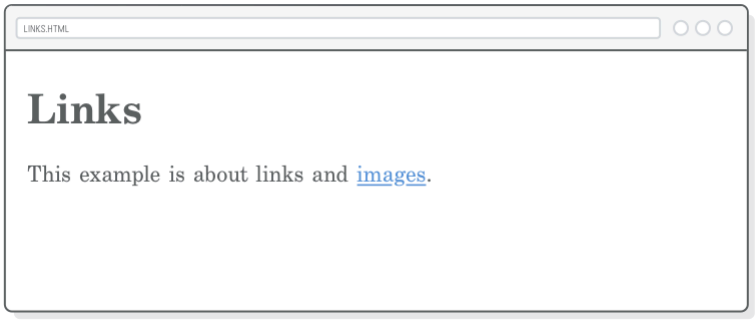


Different elements allow different attributes, and you can refer to MDN for details about which elements accept which attributes. Right now, we're concerned with the `href` attribute because it determines where the user goes when they click an `<a>` element. Update your link to match the following:

```
<p>This example is about links and <a href='images.html'>images</a>.</p>
```

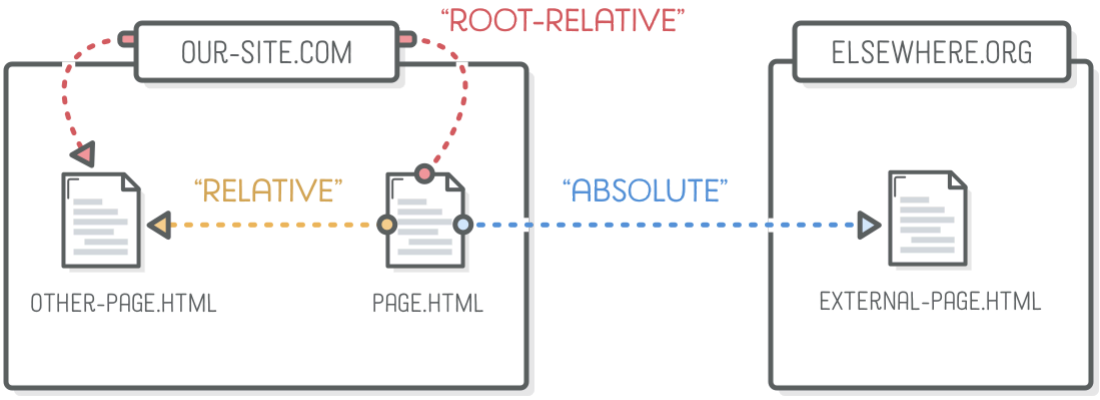
Notice how attributes live inside the *opening* tag. The attribute name comes first, then an equal sign, then the "value" of the attribute in either single or double quotation marks. This syntax distinguishes attributes from content (which goes between the tags).

The extra bit of information provided by the `href` attribute tells the browser that this `<a>` element is in fact a link, and it should render the content in its default blue text:



ABSOLUTE, RELATIVE, AND ROOT-RELATIVE LINKS

~~structured. For our purposes, a website is just a collection of HTML files~~ organized into folders. To refer to those files from inside of another file, the Internet uses "uniform resource locators" (URLs). Depending on what you're referring to, URLs can take different forms. The three types of URLs we'll be dealing with are highlighted below:



Absolute, relative, and root-relative links refer to the value of the `href` attribute. The next few sections will explain how and when to use each of them. But first, let's add the following content to our `links.html` file:

```
<p>This particular page is about links! There are three kinds of links:</p>

<ul>
  <!-- Add <li> elements here -->
</ul>
```

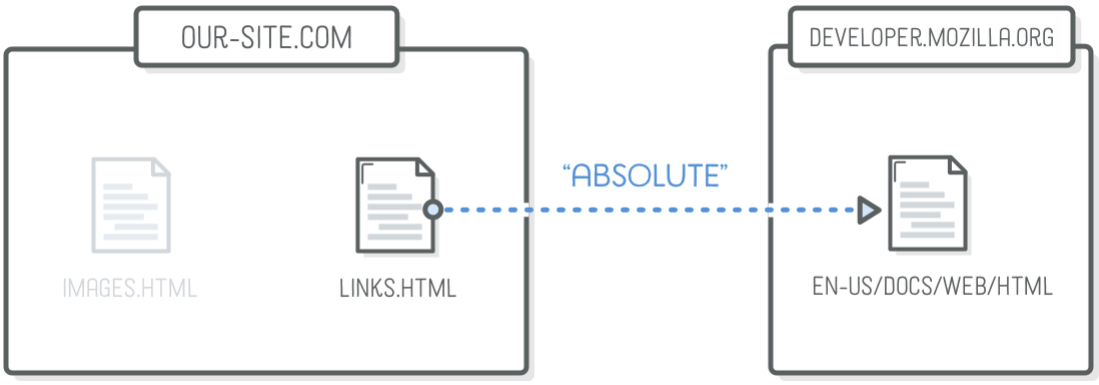
ABSOLUTE LINKS

"Absolute" links are the most detailed way you can refer to a web resource. They start with the "scheme" (typically `http://` or `https://`), followed by the domain name of the website, then the path of the target web page.

For example, try creating a link to the Mozilla Developer Network's HTML element reference:

```
<li>Absolute links, like to
    <a href='https://developer.mozilla.org/en-US/docs/Web/HTML'>Mozilla
```

It's possible to use absolute links to refer to pages in your own website, but hard-coding your domain name everywhere can make for some tricky situations. It's usually a better idea to reserve absolute links only for directing users to a different website.



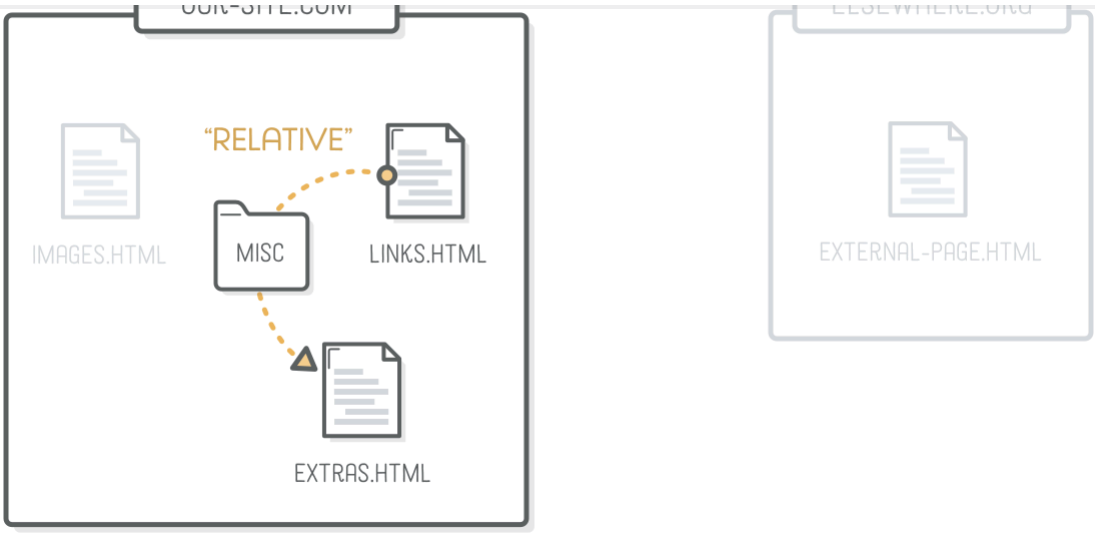
RELATIVE LINKS

"Relative" links point to another file in your website from the vantage point of the file you're editing. It's implied that the scheme and domain name are the same as the current page, so the only thing you need to supply is the path.

Here's how we can link to our `extras.html` file from inside of `links.html`:

```
<li>Relative links, like to our <a href='misc/extras.html'>extras
    page</a>.</li>
```

In this case, the `href` attribute represents the file path to `extras.html` from the `links.html` file. Since `extras.html` isn't in the same folder as `links.html`, we need to include the `misc` folder in the URL.



Each folder and file in a path is separated by a forward slash (/). So, if we were trying to get to a file that was *two* folders deep, we'd need a URL like this:

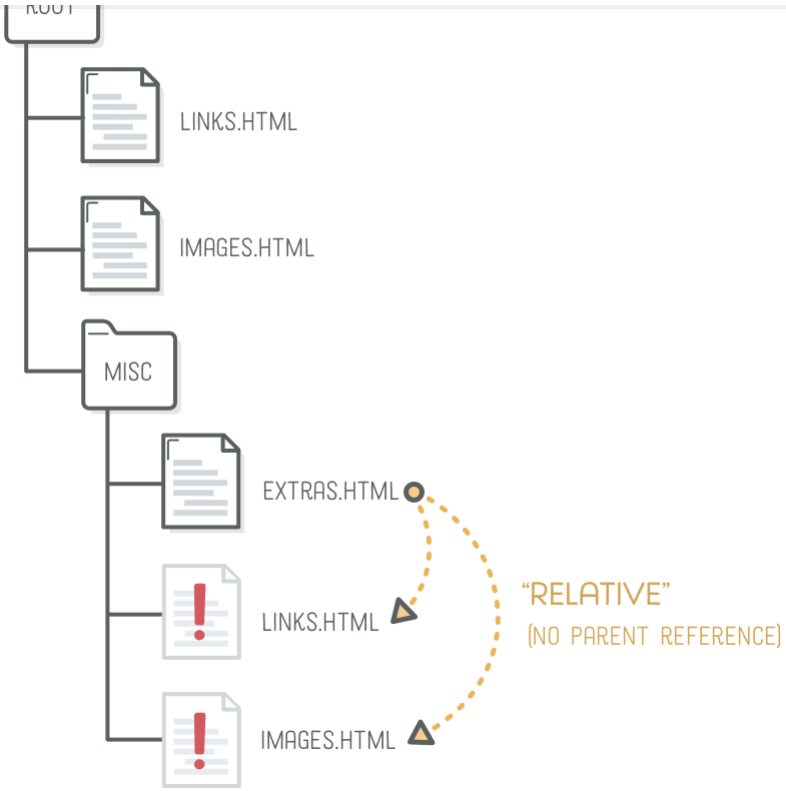
```
misc/other-folder/extras.html
```

PARENT FOLDERS

That works for referring to files that are in the same folder or a deeper folder. What about linking to pages that are in a directory *above* the current file? Let's try creating relative links to `links.html` and `images.html` from within our `extras.html` file. Add this to `extras.html`:

```
<p>This page is about miscellaneous HTML things, but you may
   also be interested in <a href='links.html'>links</a> or
   <a href='images.html'>images</a>.</p>
```

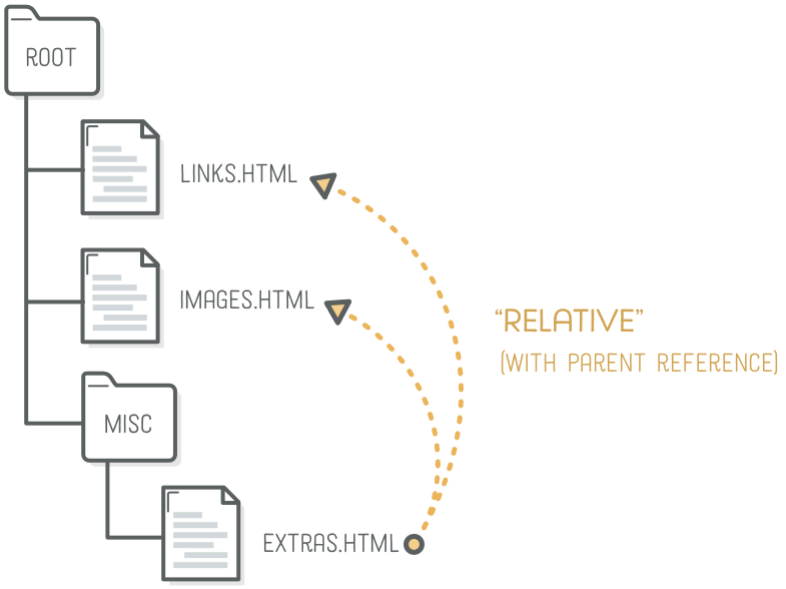
When you click either of those links in a web browser, it will complain that the page doesn't exist. Examining the address bar, you'll discover that the browser is trying to load `misc/links.html` and `misc/images.html`—it's looking in the wrong folder! That's because our links are *relative* to the location of `extras.html`, which lives in the `misc` folder.



To fix this, we need the `..` syntax. Two consecutive dots in a file path represent a pointer to the parent directory:

```
<p>This page is about miscellaneous HTML things, but you may also be
   interested in <a href='../links.html'>links</a> or <a
   href='../images.html'>images</a>.</p>
```

This is like saying, "I know `extras.html` is in the `misc` folder. Go up a folder and look for `links.html` and `images.html` in there."



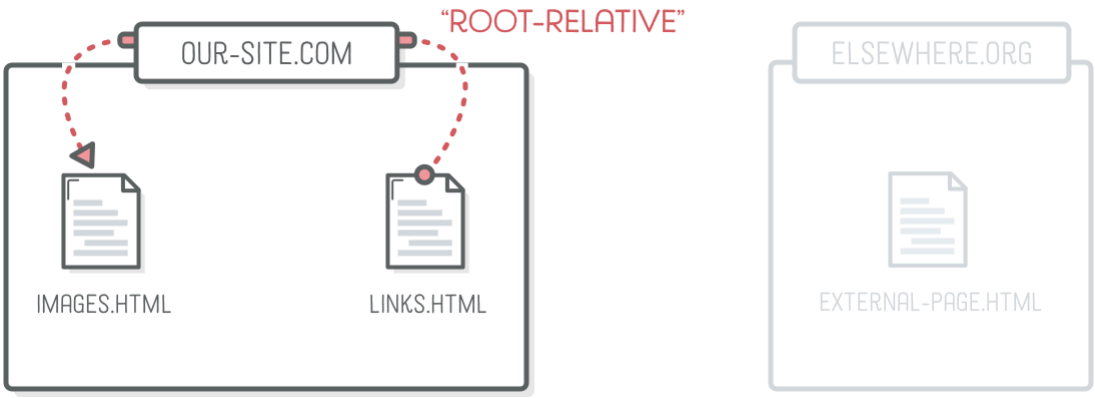
```
../../elsewhere.html
```

Relative links are nice because they let you move around entire folders without having to update all the `href`'s on your `<a>` elements, but they can get a little confusing when all your links start with a bunch of dots. They're best for referring to resources in the same folder or in a standalone section of your website.

For example, all the images in *HTML & CSS Is Hard* are loaded with relative URLs (we'll learn how to do images a moment). This allows us to rename any of our chapter slugs without having to update all our image paths.

ROOT-RELATIVE LINKS

"Root-relative" links are similar to the previous section, but instead of being relative to the current page, they're relative to the "root" of the entire website. For instance, if your website is hosted on `our-site.com`, all root-relative URLs will be relative to `our-site.com`.



Unfortunately, there is one caveat to our discussion of root-relative links: this entire tutorial uses local HTML files instead of a website hosted on a web server. This means we won't be able to experiment with root-relative links. But, if we did have a real server, the link to our home page would look like this:

```
<!-- This won't work for our local HTML files -->
<li>Root-relative links, like to the <a href='/'>home page</a> of our website,
    but those aren't useful to us right now.</li>
```

former *starts* with a forward slash. That initial forward slash represents the root of your site. You can add more folders and files to the path after that initial slash, just like relative links. The following path will work correctly no matter where the current page is located (even in `misc/extras.html`):

```
/images.html
```

Root-relative links are some of the most useful kinds of links. They're explicit enough to avoid the potential confusion of relative links, but they're not overly explicit like absolute links. You'll see a lot of them throughout your web development career, especially in larger websites where it's hard to keep track of relative references.

LINK TARGETS

Attributes alter the meaning of HTML elements, and sometimes you need to modify more than one aspect of an element. For example, `<a>` elements also accept a `target` attribute that defines where to display the page when the user clicks the link. By default, most browsers replace the current page with the new one. We can use the `target` attribute to ask the browser to open a link in a new window/tab.

Try changing our absolute link in `links.html` to match the following. Notice how the second attribute looks just like the first, but they're separated from each other by a space (or a newline):

```
<li>Absolute links, like to
    <a href='https://developer.mozilla.org/en-US/docs/Web/HTML'
       target='_blank'>Mozilla Developer Network</a>, which is a very good
       resource for web developers.</li>
```

The `target` attribute has a few pre-defined values that carry special meaning for web browsers, but the most common one is `_blank`, which specifies a new tab or window. You can read about the rest on MDN.

NAMING CONVENTIONS

You'll notice that none of our files or folders have spaces in their names. That's on purpose. Spaces in URLs require special handling and should be avoided at

and-images project called `spaces are bad.html`. Add a little bit of text to it, then open it in Google Chrome or Safari (Firefox cheats and preserves the spaces).

```
links-and-images/spaces%20are%20bad.html
```

In the address bar, you'll see that all our spaces have been replaced with `%20`, as shown above. Spaces aren't allowed in URLs, and that's the special encoding used to represent them. Instead of a space, you should always use a hyphen, as we've been doing throughout this tutorial. It's also a good idea to use all lowercase characters for consistency.

Notice how there's a direct connection between our file/folder names and the URL for the web page they represent. The names of our folders and files determine the slugs for our web pages. They're visible to the user, which means you should put in as much effort into naming your files as you put into creating the content they contain.

Estas convenciones de nomenclatura se aplican a *todos* los archivos de su sitio, no solo a los archivos HTML. Los archivos CSS, los archivos JavaScript y las imágenes también deben evitar los espacios y tener un uso uniforme de las mayúsculas y minúsculas.

IMÁGENES

A diferencia de todos los elementos HTML que hemos encontrado hasta ahora, el contenido de las imágenes se define *fuera* de la página web que las muestra. Afortunadamente para nosotros, ya tenemos una forma de hacer referencia a recursos externos desde dentro de un documento HTML: URL absolutas, relativas y relativas a la raíz.

Las imágenes se incluyen en las páginas web con la `<img/>`etiqueta y su `src`atributo, que apunta al archivo de imagen que desea mostrar. Observe que es un elemento vacío, como `<br/>`y `<hr/>`del capítulo anterior. (No agregue esto a nuestro proyecto todavía. Trataremos ejemplos concretos en la siguiente sección).

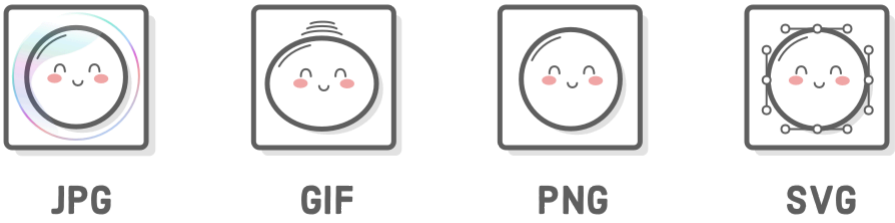
```
<img src='some-photo.jpg'/>
```

sea un poco más complicado que con una etiqueta simple `<img/>`. Dejaremos estas complejidades para el capítulo Imágenes adaptables de este tutorial. Asegúrese también de consultar el elemento `<figure>`and en el capítulo HTML semántico .`<figcaption>`

Por ahora, centrémonos en los numerosos formatos de imágenes que circulan por Internet.

FORMATOS DE IMAGEN

En la Web se utilizan cuatro formatos de imagen principales, y todos ellos fueron diseñados para distintas funciones. Comprender su finalidad es de gran ayuda para mejorar la calidad de las páginas web.



En las siguientes secciones, analizaremos casos de uso ideales para cada uno de estos formatos de imagen. Asegúrese de haber descomprimido estas imágenes mochi en su `links-and-images`proyecto antes de continuar.

IMÁGENES JPG

Las imágenes JPG están diseñadas para manejar paletas de colores grandes sin aumentar de forma exorbitante el tamaño del archivo. Esto las hace ideales para fotos e imágenes con muchos degradados. Por otro lado, los JPG no permiten píxeles transparentes, lo que se puede ver en los bordes blancos de la imagen siguiente si se observa con atención:

Incruste esta `mochi.jpg`imagen en nuestra `images.html`página con el siguiente fragmento (esto también incluye un poco de navegación a nuestras otras

```
<p>This page covers common image formats, but you may also be looking for <a
   href='links.html'>links</a> and <a href='misc/extras.html'>useful
   extras</a>.</p>

<h2>JPGs</h2>

<p>JPG images are good for photos.</p>

<img src='images/mochi.jpg'/>
```

IMÁGENES GIF

Los GIF son la opción preferida para las animaciones simples, pero la desventaja es que son algo limitados en términos de paleta de colores; nunca los uses para fotos. Los píxeles transparentes son una opción binaria para los GIF, lo que significa que no puedes tener píxeles semiopacos. Esto puede dificultar la obtención de altos niveles de detalle en un fondo transparente. Por este motivo, generalmente es mejor usar imágenes PNG si no necesitas animación.

Puedes agregar a este pequeño a nuestro `images.html`archivo con lo siguiente:

```
<h2>GIFs</h2>

<p>GIFs are good for animations.</p>

<img src='images/mochi.gif'/>
```

IMÁGENES PNG

PNGs are great for anything that's not a photo or animated. For photos, a PNG file of the same quality (as perceived the human eye) would generally be bigger than an equivalent JPG file. However, they do deal with opacity just fine, and they don't have color palette limitations. This makes them an excellent fit for icons, technical diagrams, logos, etc.

Let's add this PNG image to our example project as well:

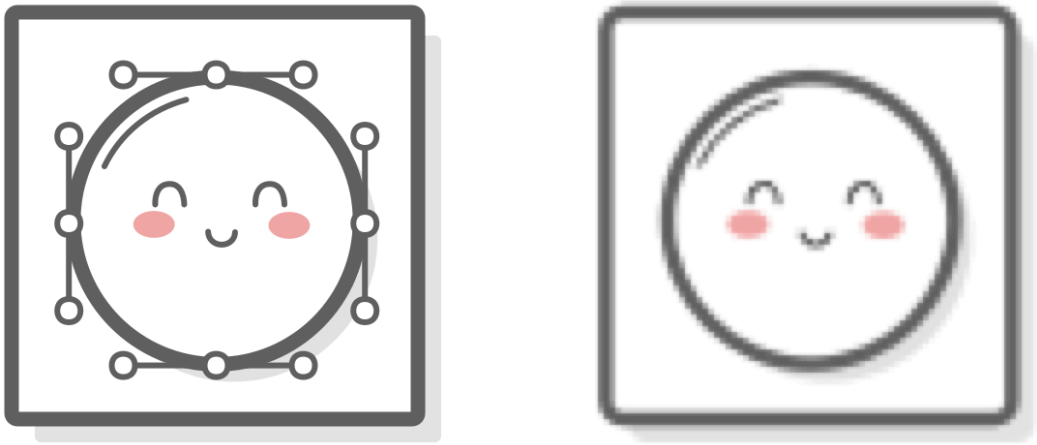
```
<h2>PNGs</h2>

<p>PNGs are good for diagrams and icons.</p>

<img src='images/mochi.png'/>
```

SVG IMAGES

Unlike the pixel-based image formats above, SVG is a vector-based graphics format, meaning it can scale up or down to *any* dimension without loss of quality. This property makes SVG images a wonderful tool for responsive design. They're good for pretty much all the same use cases as PNGs, and you should use them whenever you can.



These 300×300 pixel images were originally 100×100 pixels, but scaling them up clearly shows the difference between them. Notice the crisp, clean lines on the left SVG image, while the PNG image on the right is now very pixelated.

Despite being a vector format, SVGs can be used exactly like their raster counterparts. Go ahead and add the `mochi.svg` to our `images.html` page:

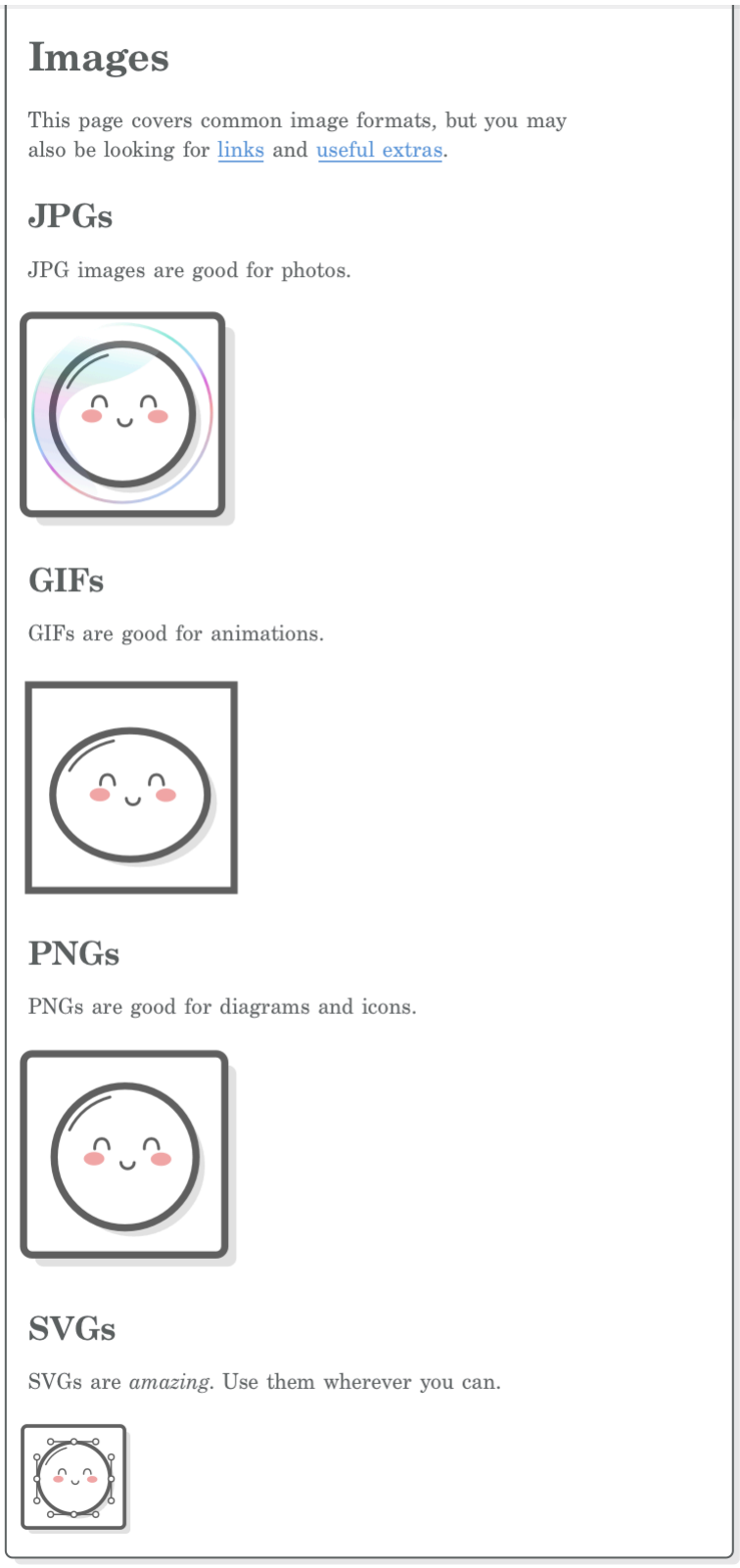
```
<h2>SVGs</h2>

<p>SVGs are <em>amazing</em>. Use them wherever you can.</p>
```

There is one potential issue with SVGs: for them to display consistently across browsers, you need to convert any text fields to outlines using your image editor (e.g., Abode Illustrator or Sketch). If your images contain a lot of text (like the fancy screenshots in this tutorial), this can have a big impact on file size. For this reason, *InternetingIsHard.com* uses PNGs instead of SVGs, even though SVGs are so awesome.

IMAGE DIMENSIONS

By default, the `<img/>` element uses the inherit dimensions of its image file. Our JPG, GIF, and PNG images are actually 150×150 pixels, while our SVG mochi is only 75×75 pixels.



As we'll discuss further in Responsive Images, pixel-based image formats need to be twice as big as you want them to appear on retina displays. To get our pixel-based images down to the intended size (75×75), we can use the `<img/>` element's `width` attribute. In `images.html`, update all of our pixel-based images to match the following:

```
<!-- In GIFs section -->
<img src='images/mochi.gif' width='75'/>

<!-- In PNGs section -->
<img src='images/mochi.png' width='75'/>
```

The `width` attribute sets an explicit dimension for the image. There's a corresponding `height` attribute, as well. Setting only one of them will cause the image to scale proportionally, while defining both will stretch the image. Dimension values are specified in pixels, and you should never include a unit (e.g., `width='75px'` would be incorrect).

The `width` and `height` attributes can be useful, but it's usually better to set image dimensions with CSS so you can alter them with media queries. We'll discuss this in more detail once we get to responsive design.

TEXT ALTERNATIVES

Adding `alt` attributes to your `<img/>` elements is a best practice. It defines a "text alternative" to the image being displayed. This has an impact on both search engines and users with text-only browsers (e.g., people that use text-to-speech software due to a vision impairment).

Update all our images to include descriptive `alt` attributes:

```
<!-- In JPGs section -->
<img src='images/mochi.jpg' width='75' alt='A mochi ball in a bubble'/>

<!-- In GIFs section -->
<img src='images/mochi.gif' width='75' alt='A dancing mochi ball'/>

<!-- In PNGs section -->
<img src='images/mochi.png' width='75' alt='A mochi ball'/>

<!-- In SVGs section -->
<img src='images/mochi.svg' alt='A mochi ball with Bézier handles'/>
```

For more examples of how to use the `alt` attribute, please refer to the ridiculously detailed official specification.

Now that we're (hopefully) more than comfortable with the HTML attribute syntax, we can add a couple of finishing touches to our HTML boilerplate code. Every web page you create should define the language it's written in and its character set.

DOCUMENT LANGUAGE

A web page's default language is defined by the `lang` attribute on the top-level `<html>` element. Our document is in English, so we'll use the `en` country code as the attribute value (do this for *all* of the pages we created):

```
<html lang='en'>
```

If you're not sure what the country code for your language is, you can look it up here under the **Subtag** field.

CHARACTER SETS

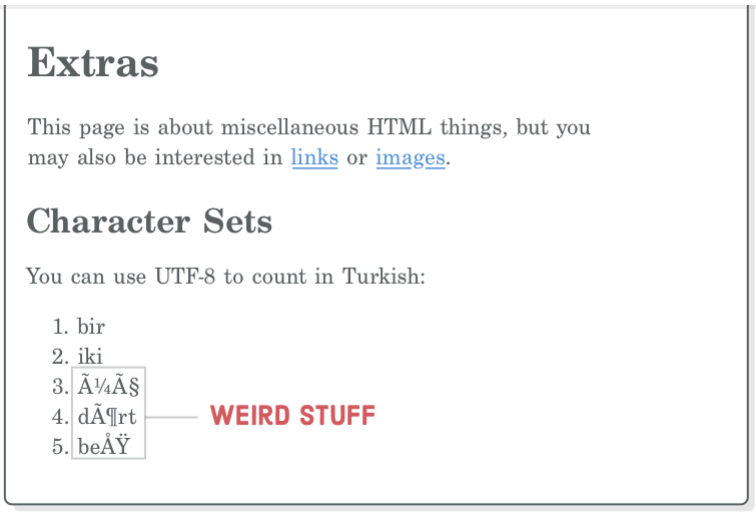
A "character set" is kind of like a digital alphabet for your browser. It's different from the language of your document in that it only affects how the letters themselves are rendered, not the language of the content. Let's copy and paste some international characters into our `misc/extras.html` web page and see what happens.

```
<h2>Character Sets</h2>

<p>You can use UTF-8 to count in Turkish:</p>

<ol>
  <li>bir</li>
  <li>iki</li>
  <li>üç</li>
  <li>dört</li>
  <li>beş</li>
</ol>
```

When you view this in a browser, you'll see some weird stuff where the ü, ç, ö, and ş characters should be:



That's because the default character set for most browsers doesn't accommodate these "special" characters. To fix this, specify a UTF-8 character encoding by adding a `<meta>` element with a `charset` attribute to the `<head>` of our `misc/extras.html` file:

```
<meta charset='UTF-8'/>
```

The special characters should now render correctly. These days, UTF-8 is sort of like a universal alphabet for the Internet. Every web page you create should have this line in its `<head>`.

HTML ENTITIES

Ok, so this last section doesn't actually have anything to do with links or images, but we do need to discuss one more thing before switching gears into CSS. An "HTML entity" is a special character that can't be represented as plain text in an HTML document. This typically either means it's a reserved character in HTML or you don't have a key on your keyboard for it.

RESERVED CHARACTERS

Los caracteres `<`, `>`, y `&`se denominan "caracteres reservados" porque no se permite insertarlos en un documento HTML sin codificarlos. Esto se debe a que significan algo en la sintaxis HTML: `<`comienzan una nueva etiqueta, `>`terminan una etiqueta y, como veremos a continuación, `&`inician una entidad HTML.

En `misc/extras.html`, agregue lo siguiente:

```
should always use HTML entities for these three characters.</p>
```

Las entidades siempre comienzan con un ampersand (`&`) y terminan con un punto y coma (`;`). Entre ellos, se coloca un código especial que el navegador interpretará como un símbolo. En este caso, interpreta `lt`, `gt`, y `amp`como símbolos de menor que, mayor que y ampersand, respectivamente.

Hay un montón de entidades HTML. Te dejaremos que explores la mayoría de ellas por tu cuenta.

CITAS

Las comillas curvas no son necesarias, pero si te preocupa la tipografía, serán unas de las entidades HTML más comunes que utilizarás. Hay cuatro tipos diferentes de comillas curvas (comillas simples y dobles de apertura y cierre):

- `&ldquo;`
- `&rdquo;`
- `&lsquo;`
- `&rsquo;`

Puedes utilizarlas en lugar de comillas `'`simples `"`, de la siguiente manera:

```
<p>If you&rsquo;re into &ldquo;web typography,&rdquo; you&rsquo;ll also find
   yourself using curly quotes quite a bit.</p>
```

A diferencia de las comillas rectas, estas entidades de comillas curvas deben abrazar el texto.

ENTIDADES UTF-8 Y HTML

En los viejos tiempos de la web, no se permitía que los archivos HTML tuvieran caracteres especiales, lo que hacía que las entidades fueran mucho más útiles. Pero, como ahora utilizamos un conjunto de caracteres UTF-8, deberíamos poder insertar cualquier carácter directamente en el documento HTML. Esto hace que las entidades sean útiles principalmente como caracteres reservados o por conveniencia al crear HTML puro.

Un sitio web es básicamente un conjunto de archivos HTML, imágenes y (como veremos en breve) archivos CSS vinculados entre sí. Deberías empezar a pensar en un sitio web como una forma atractiva para que los usuarios naveguen por las carpetas y los archivos que creamos como parte del proceso de desarrollo web. Con esa perspectiva, debería quedar claro que mantener un sistema de archivos bien organizado es un aspecto fundamental de la creación de un sitio web.

También aprendimos sobre algunos atributos importantes (`lang`y `charset`) que nos brindan la plantilla básica que debes usar como comienzo de cada página web que crees:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Some Web Page</title>
  </head>
  <body>
    <h1>Some Web Page</h1>
    <!-- Rest of the page content -->
  </body>
</html>
```

Sin embargo, todavía nos falta una pieza muy importante: CSS. En el próximo capítulo, descubriremos más elementos y atributos HTML que nos permitirán adjuntar estilos CSS a todo nuestro sitio web. La capacidad de trabajar con varios archivos y vincularlos entre sí de forma inteligente será aún más importante que en este capítulo.

SIGUIENTE CAPÍTULO >

de los lectores, así que ¡ ven a saludarlo !

Próximamente habrá más tutoriales (muchos más).
Ingresa tu correo electrónico y te avisaremos cuando estén disponibles.

TU@EJEMPLO.COM

SUSCRIBIR