HTML Anchors: Here's How To Create Links For Fast Navigation

Last Updated on November 6, 2019

The anchor element is used to create hyperlinks between a *source anchor* and a *destination anchor*. The source is the text, image, or button that links to another resource and the destination is the resource that the source anchor links to.

Hyperlinks are one of the fundamental technologies that make the web the information superhighway, and understanding how to use anchor elements is one of the first things you need to master when learning HTML.

Contents [hide]

The Anchor Element

The anchor element tag is the letter "a" surrounded by angle brackets like this: `<a>` . Both the opening and closing attributes are required, and all of the content between the tags makes up the anchor source.

```
If we want to use just a single word as an anchor, we
wrap just that one <a>word</a> in anchor tags.
```

```
We could also wrap an entire paragraph in anchor
tags if we wanted the entire paragraph to link
somewhere.

<!--We can also use an image as an anchor element-->
<a><img src="http://example.com/image-file.png" /></a>
```

While the examples above will produce anchor elements they aren't of much use since we haven't included any additional instructions. Right now these anchor elements link to nothing. To link a source anchor to a destination anchor, we need to apply some additional attributes to the anchor element.

In this short tutorial we'll cover the attributes you can use to add a destination anchor to your hyperlinks, tell the browser what to do with the link, and add semantic meaning to anchor elements for browsers and web crawlers to use.

The Most Important Anchor Attributes

There are three anchor attributes you need to know to create functional hyperlinks. These attributes are `href`, `target`, and `download`.

Specify a Hyperlink Target: `href`

The hypertext reference, or `href`, attribute is used to specify a target or destination for the anchor element. It is most commonly used to define a URL where the anchor element should link to.

```
In this example, the <a href="http://example.com">anchored text</a>
links to the URL <em>www.example.com</em>.
```

An `href` can do a lot more than just link to another website.

- It can be used to link directly to any element on a web page that has been assigned an `id`.
- It can be used to link to a resource using a protocol other than http.
- It can be used to run a script.

The following HTML includes examples of all three of these actions:

```
<a href="#Specify_a_Hyperlink_Target_href">This first anchor element</a> links back to the heading
of this section of the tutorial by linking to an <code>id</code> we
assigned to the section heading element.

We can use the <code>mailto</code> protocol to create a link that will
try to launch the default email program on your computer. Let's give it
a try! We'd love for you to <a href="mailto:contact@html.com">
get in touch</a> with us!

Lastly, if you click the link below a complimentary JavaScript alert
window will appear. <a>
Click the link to see what happens</a>.
```

In the box below you'll see that HTML as it renders in the browser. Try each link to see what happens.

This first anchor element links back to the heading of this section of the tutorial by linking to an `id` we assigned to that heading element.

We can use the `mailto` protocol to create a link that will try to launch the default email program on your computer. Let's give it a try! We'd love for you to get in touch with us!

Lastly, if you click the link below a complimentary JavaScript alert window will appear. Click the link to see what happens.

The `href` element is pretty easy to use, but we can get a lot more mileage out of our anchor elements by getting to know the `target` attribute.

Specify a Location to Open the Link: `target`

Let's look back at the `mailto` link we created in the previous example. Here's that code again:

```
We'd love for you to <a href="mailto:contact@html.com">
get in touch</a> with us!
```

While that link is useful, the way it opens isn't ideal. Depending on how your computer and browser are configured, it's possible that clicking on the link created by that text will redirect your browser window away from this website to a web-based email application such as Gmail or Yahoo! Mail. While we certainly want to encourage our visitors to contact us, we don't want to do so at the expense of sending visitors away from our website.

The answer to this conundrum is the `target` attribute paired with the `_blank` value.

```
We'd love for you to <a href="mailto:contact@html.com" target="_blank" rel="noopener">get in touch</a> with u
s!
```

When we add the `target` attribute and `_blank` attribute to our link we tell the visitors browser to open the link in a new (blank) browser tab or window. If we render that code in the browser, when a visitor clicks the link it will open in a new tab.

We'd love for you to get in touch with us!

Specify a Resource to Download: `download`

Links are also used to tell a browser to start downloading a file. The `download` attribute is used to identify a link that should initiate a download and the value assigned to the `download` attribute is the name of the file to be downloaded.

The `href` attribute also comes into play when setting up an anchor element that initiates a download. While the `download` attribute names the file, the `href` attribute points to the location where the file is hosted.

Here's an example of how a download-initiating anchor element would look:

```
To create a link that tells a browser to <a href="http://example.com/file.ext" download="Example_File">downloa
d a file</a>, use the
<code>href</code> attribute to identify the file to be
downloaded, and the <code>download</code> attribute to
provide a name for the downloaded file.
```

If the `href` in the example above pointed to a real file, when the download was complete the downloaded file would be called *Example_File.ext* where *.ext* would be the format of the file that was downloaded. It isn't necessary to include the file type extension in the value assigned to the `download` attribute. The file extension will be automatically identified when the browser downloads the file, and automatically appended to the downloaded file name.

Internal and External Links

There are two types of resources we can link to using the `a` element: internal and external. Internal links are those that point to other pages of our website. External links point to web pages that aren't part of our website.

Building internal links is important for a few different reasons:

- Internal links are used to create navigation menus that help website visitors navigate our website.
- Internal links are used in the text of website content to help website visitors locate related content.
- Internal links are also used by search engine web crawlers to locate the pages of a website and to share authority (also known as link juice) with the other pages of a website.

When writing internal links make sure you don't overdo it. The link juice of any given web page is shared between the links on the page. So the more links you put on a page, the more diluted the juice passed to each link becomes. A good rule of thumb is to have no more than 100 links per page although there are exceptions in the case of extremely large and complex websites.

External links are also important for a few different reasons:

- External links may be recommended, required, or just best-practice to provide proper attribution to the source of an idea or a resource.
- External links allow us to refer website visitors to useful related content.
- When other websites post external links that point at our website, these links known as backlinks allow link juice to flow to our website and improve our website's position on search engine results pages (SERP).

When writing external links try to avoid linking to direct competitors. You don't want to help their SERP ranking for search terms that you are targeting for your own website. It's also a good idea to use the `target="_blank"` attribute when writing external links. By opening external links in a new tab you keep visitors on your site for longer.

Never Say "Click Here"

Surely you've seen and maybe even created links that look something like this:

```
To learn more <a href="http://example.com">click here</a>!
```

There are at least three good reasons why using generic anchor element text such as "Click Here" is a terrible practice.

- Website visitors who depend on assistive technologies such as screen readers will have a hard time deciphering the meaning of links that make use of generic link text such as "Click Here".

- Website visitors who are scanning a page will have to take several additional seconds to investigate the text around the link to have an idea of what the anchor links to.
- Search engine web crawlers associate anchor element text with the link URL. Properly selected anchor element text helps search engines determine the relevance of a web page to specific keywords. Generic text tells search engines nothing about the linked web page.

Ideal anchor element text is succinct and identifies a keyword or keywords that are relevant to the web page. Here's a sentence that includes two examples of properly selected link text:

```
If you want your website to get more traffic from
<a href="http://www.google.com/" target="_blank" rel="noopener">search engines</a>,
check out our <a href="/seo/">SEO tutorial</a> for tips on improving
your website's SERP ranking.
```

The first link tells web crawlers and website visitors that the website `http://www.google.com` has something to do with the keywords "search engines". The second tells visitors and search engines that the page found at the relative URL `/seo/` is related to the keywords "SEO tutorial".

Absolute vs. Relative URLs

There were two links in our last example: `http://www.google.com` and `/seo/`. The first is an example of an *absolute URL*. Absolute URLs are those that include a complete (absolute) description of the link destination. Absolute URLs include the protocol (*http*) and the complete domain name and file path needed to reach the destination anchor.

The second is an example of a *relative URL*. Relative URLs link to a web page by describing the position of the page *relative* to the position of the current page. When writing internal links that point to other pages of the same website we have the option of writing relative URLs rather than absolute URLs.

For example, if we are at this address: `http://example.com/products/red-product/`, and we want to link to the blue product page, we can use either of the following anchor elements:

```
<a href="/products/blue-product/">Go check out the blue product!</a>
<a href="../blue-product/">Go check out the blue product!</a>
```

What the first syntax says is: "look in the lowest level of the file directory for the file products, and look for blue products within that file". What the second syntax says is: "look in the parent folder of the `red-product` directory for a directory with the name `blue-product`. Both of these anchor elements point to the URL: `http://example.com/products/blue-product/`.

Let's say that we want to link to *About Us* – a page that exists as a direct descendant of the homepage. Here's how we could do that:

```
<a href="/about-us/">Learn more about us.</a>
```

That syntax says "go back to the lowest level of the file directory tree and then look for a directory titled *about-us*".

As a final example, let's say we are at this address once again: `http://example.com/products/red-product/`, and that we want to link to a subcategory page that exists within the `/red-products/` category.

```
<a href="large.php">See all Red Products available in Large!</a>
```

What that relative URL will do is tell the browser to look in the current folder for a file named `large.php` and will resolve to this URL: `http://example.com/products/red-product/large.php`.

To summarize, when we write relative URLs we have three options:

- When we start a relative URL with "/" we tell the browser to look for the specified file or folder in the lowest level of the file directory.
- If we start a relative URL with "../" we tell the browser back up in the directory tree one level.
- If we start a relative URL with a file or folder name we tell the browser to look for the specified resources in the current file.

Why Are Relative URLs Used?

Relative URLs are very popular with web developers. It is common for websites to be developed on a staging server with a domain name that is not the same as the eventual permanent domain name. If relative URLs are used, when the site is transferred from the staging server to being live on the web all of the relative URLs will continue to work just fine. However, if absolute URLs are used the development team will have to go through the site fixing all of the URLs.

While the benefit of using relative URLs during the development of a site on a staging server not inconsiderable, it comes at a cost.

While you should know how relative URLs work, many experienced developers and SEO experts recommend sticking to absolute URLs as much as possible.

Other Useful Anchor Attributes

With just the three attributes we've covered so far, you can complete every hyperlinking task you will encounter on the web. However, there are additional attributes that can be used to tell a visitor's web browser and search engines that index our website something about the meaning of the hyperlinks.

Specify the Language of the Anchor Destination: `hreflang`

This is optional tag can be used to tell a web browser the language of the document to which the anchor is linked. For example, if we were linking to a URL and wanted to tell the browser that the destination anchor was in Spanish we would use the appropriate language code in ISO 639-1 format as the value of the `hreflang` attribute.

```
To tell the browser that a <a href="http://example.com" hreflang="es">a link</a> points to a resource that is
 in a
different langauge, we can use the <code>hreflang</code>
attribute.
```

Specify the Relationship Between Source and Destination: `rel`

The relationship, or `rel` , attribute is used to describe the relationship between the source anchor and the destination anchor. For example, we could use the following code to identify the destination as being a resource about the author of the document.

```
This artile was written by <a href="https://plus.google.com/+AuthorName" rel="author">Great Author</a>.
```

There are several different values that can be used to describe the relationship between the source link and the destination link. Here are a few of the most common:

- `rel="nofollow"` : Use this attribute if you're linking to a resource you don't want to lend your site's reputation to. For example, if you link to a spammy site as an example of what *not* to do, you would want to use this value to keep from contributing to the site's search engine visibility.
- `rel="alternate"` : If your site has more than one version – such as a translation into a different language – use this value to identify the linked resource as an alternate version of the current page.
- `rel="bookmark"` : This value is used to identify a URL formulation that is permanent and may be used for bookmarking.

- `rel="help"` : Identify a linked resource as a help file for the current page with this value.
- `rel="license"` : Use this attribute when linking to a recognized copyright license.
- `rel="next"` : If the current document exists in a series of documents, use this value when linking to the next document in the series.
- `rel="prev"` : If the current document exists in a series of documents, use this value when linking to the previous document in the series.
- `rel="noreferrer"` : If you want to link to an external website but avoid letting the destination website know who the referrer is, use this value.

Specify the Internet Media Type: `type`

The `type` attribute is an optional attribute that can be used to identify the Internet media type, or MIME type, of the destination document. For example, if linking to a website rendered in HTML you could add the attribute `type="text/html"` to tell a visitor's browser that the link points to an HTML document.

Right now this attribute doesn't do much of anything, but the thinking is that in the future the information contained in this attribute could be used to somehow communicate to a website visitor the type of content they are about to be linked to before clicking on the link.

The `type` value must be a valid IANA media type. Some of the most commonly used values are: `text/html` , `text/css` , `application/javascript` , and `multipart/form-data` .

A Web of Links

The World Wide Web is best described as a massive library of hyperlinked documents where anchor elements are used to create bridges between related documents. In this definition, anchor elements occupy their rightful place as the glue that ties the web together and the bridges that allow web users to move from one document to a related document.

One common way for links to be used is to build a navigation menu that lays out the logical, hierarchical structure of a website. If you want to learn how to build effective navigational menus read our tutorial on that topic.

Jon Penland

Jon is a freelance writer, travel enthusiast, husband and father. He writes about web technologies such as WordPress, HTML, and CSS.

Related Elements

Element Name	Attributes	Notes
link	href rel media title type	The <link> element is used to define a relationship between an HTML document and an external resource. This element is most commonly used to define the relationship between a document and one or more external CSS stylesheets.
anchor	hreflang download	The <a> element, or anchor element, it used to create a hyperlink to

	target title href name	another webpage or another location within the same webpage. The hyperlink created by an anchor element is applied to the text, image, or other HTML content nested between the opening and closing <a> tags.
base	target href	The <base> element is used to identify a base URL upon which to build all relative URLs that appear on a webpage. In addition, if the <base> element has a target attribute, the target attribute will be used as the default attribute for all hyperlinks appearing in the document.

Tutorials and Resources

How to Create Responsive Menus with CSS

Using Character Codes In Your HTML? Get This Quick Reference Sheet Now

Last Updated on July 9, 2019

HTML character references are short bits of HTML, commonly referred to as character entities or entity codes, that are used to display characters that have special meaning in HTML as well as characters that don't appear on your keyboard.

- Characters with special meaning in HTML are called reserved characters. For example, left (<) and right (>) angle brackets are reserved in HTML to identify the opening and closing tags of elements.
- Characters that don't appear on your keyboard include things like the copyright symbol (©) and the mathematical value pi (π).

If we want to use these types of characters in an HTML document and have them appear when rendered in a browser we use HTML character references.

Contents [hide]

A Practical Example

Let's say that you want to display a block of HTML in a web page and have the element tags show up on the page. You may try to do so by simply dropping `<code>` blocks around the block of HTML you want to display. However, what you will find is that even with the `<code>` tags surrounding the bit of HTML in question, it will still be processed as HTML and rendered by the browser. What we can do is replace all of the special characters with the appropriate character references to prevent the browser from processing the code.

```
<!--The <code> blocks don't prevent this HTML from being rendered-->
<code>
  <p>This is a list of items.</p>
  <ul>
    <li>List Item A</li>
    <li>List Item B</li>
    <li>List Item C</li>
  </ul>
</code>

<!--Replace special characters with character references-->
<code>
```

```
  <p>This is a list of items.<&sol;p>
  <ul>
    <li>List Item A<&sol;li>
    <li>List Item B<&sol;li>
    <li>List Item C<&sol;li>
  <&sol;ul>
</code>
```

Let's see how that code renders in the browser.

This is a list of items.

- List Item A
- List Item B
- List Item C

```
<p>This is a list of items.<&sol;p>
<ul>
<li>List Item A<&sol;li>
<li>List Item B<&sol;li>
<li>List Item C<&sol;li>
<&sol;ul>
```

As you can see, the code blocks around the first block of code did not prevent the browser from processing the HTML. However, by replacing certain characters in the second block with HTML character references, we can display the code block as HTML markup.

Character Entity Format

In HTML, there are three different ways to format a character entity. You can use the character name, a Unicode value, or a number. For example, an ampersand may be displayed using any of the following entities: `&` , `&` , or `&` .

In all three cases, the format looks basically the same. Each entity begins with an ampersand (&), followed by the character name, Unicode, or number reference, and ends with a semicolon. When a number is used, it must be preceded by the pound symbol (#), and when a Unicode value is used, it must be preceded by a pound symbol and the letter x (#x).

Most people use character names rather than Unicode values or numbers when adding named characters to HTML documents since they're much easier to remember, but it's equally acceptable to use either the Unicode or number references as well.

Diacritics

There is one special subtype of character entity code that merits special mention: diacritical marks. These are marks that appear directly over the preceding letter and include accent marks and tildes. Here are the three most common diacritics:

Mark	Character Name	Number	Example
Acute	´	´	á produces á
Grave	`	`	à produces à
Tilde	˜	~	ã produces ã

Support for diacritical mark character names is limited right now, and you will see more consistent results between browsers if you stick with the number codes until more browsers add support for the character names.

Most Common Character Codes

Here is a quick reference table with a few of the most commonly seen HTML character references:

Symbol	Character Name	Number	Unicode	Example
Less Than	<	<	<	<
Greater Than	>	>	>	>
Slash	/	/	/	/
Quotation	"	"	"	“
Apostrophe	'	'	'	‘
Ampersand	&	&	&	&
Copyright	©	©	©	©
Registered Trademark	®	®	®	®
Degree	°	°	°	°
Left-pointing double angle	«	«	«	«
Right-pointing double angle	»	»	»	»
Non-Breaking Space				

Full List of Reserved Character Codes

A complete list of all HTML character references is maintained by the World Wide Web Consortium as part of the HTML specification.

Jon Penland

Jon is a freelance writer, travel enthusiast, husband and father. He writes about web technologies such as WordPress, HTML, and CSS.

Lists Bring Order To Web Pages: Here's The HTML Code To Create Them

Sharing is caring!

Last Updated on November 14, 2019

Lists are used all the time on the web. Articles, website navigation menus, and product features on e-commerce websites all make frequent use of lists – even when you can't tell that a list is being used just by looking at the web page.

There are three types of lists you can use, and this quick guide will show you how to use each.

Contents [hide]

Unordered Lists

An unordered list is a list in which the order of the list items does not matter. Unordered lists should be used when rearranging the order of the list items would not create confusion or change the meaning of the information on the list.

The `ul` element opens and closes an unordered list. The items on the list are contained between list item, `li` , tags. A simple unordered list containing three items could be created with the following HTML.

```
<ul>
    <li>Item A</li>
```

```
    <li>Item B</li>
    <li>Item C</li>
</ul>
```

Unless CSS rules are created to change the appearance of the list, the default presentation of an unordered list is to add a disc-style bullet point on the left-hand side of each list item and to indent the entire list.

Here's how our short unordered list renders in a browser:

- Item A
- Item B
- Item C

Ordered Lists

Ordered lists are used for lists of items for which the order of the items does matter. The syntax for an ordered list is exactly the same as for an unordered list. However, to create an ordered list, the `ol` tag is used rather than the `ul` tag. By making this one change, we can convert the unordered list in our previous example into an ordered list.

We're also going to change the text of the list items to make it clear that these are items that need to appear in a specific sequential order.

```
<ol>
    <li>Step 1</li>
    <li>Step 2</li>
    <li>Step 3</li>
</ol>
```

As you can see below, rather than a bulleted list, we now have a numbered list.

1. Step 1
2. Step 2
3. Step 3

Changing Numbering

There are times when you want to control the numbering of ordered lists. For example, your list may be broken up by a paragraph that appears mid-list to expand on a certain concept, or you may create a countdown list that begins at a high number and counts down. Lastly, maybe you'd rather use roman numerals. HTML and CSS make it easy to control the numbering of ordered lists.

Creating a Countdown List

To reverse the number of a list, simply add the `reversed` attributed to the opening `ol` tag.

```
<ol reversed>
    <li>Item 3</li>
```

```
    <li>Item 2</li>
    <li>Item 1</li>
</ol>
```

When rendered in most browsers the numbering of this list will appear reversed.

3. Item 3
2. Item 2
1. Item 1

Note that Microsoft browsers do not support the `reversed` attribute. If you use this attribute, bear in mind that visitors using Internet Explorer or Edge will see standard numbering.

Starting a List on a Specific Number

The `start` attribute is used to specify the number on which an ordered list starts. For example, imagine you have a list of 5 items, and after the second and fourth items you want to add a sentence or two with additional details. You could use the following HTML to do this without restarting the list numbering after each paragraph.

```
<ol>
    <li>Step 1</li>
    <li>Step 2</li>
</ol>
<p>A few short sentences about Item 2 that we don't want to appear appended to the list item. A second sentenc
e of additional details</p>
<ol start="3">
    <li>Step 3</li>
    <li>Step 4</li>
</ol>
<p>Notice that we used the <code>start</code> attribute on the <code>ol</code> tag to restart the numbering at
"3" following the break in the list above. We'll use the same technique to properly number Step 5 below.</p>
<ol start="5">
    <li>Step 5</li>
</ol>
```

Here's how that list renders in the browser:

1. Step 1
2. Step 2

A few short sentences about Item 2 that we don't want to appear appended to the list item. A second sentence of additional details

3. Step 3
4. Step 4

Notice that we used the `start` attribute on the `ol` tag to restart the numbering at "3" following the break in the list above. We'll use the same technique to properly number Step 5 below.

5. Step 5

Changing the Numbering Style

You can use CSS to change the marker style of an ordered list. In addition to standard numbering (referred to as `decimal` in CSS), you can also use:

- `upper-roman` for uppercaseroman numerals
- `lower-roman` for lowercase roman numerals
- `decimal-leading-zero` to add a "0" placeholder before single-digit list items

We cover the `list-style-type` CSS property used to implement these numbering styles below.

Description Lists

Description lists are created with the `dl` tag. Used far less frequently than their ordered and unordered peers, description lists are used to contain name-value groups. Each name-value group consists of one name, or term, placed between `dt` tags, followed by one or more values with each value, or description, placed between `dd` tags.

For example, if we wanted to use a description list to explain the relationship between members of a family, we might use the following code:

```
<dl>
    <dt>Parents</dt>
    <dd>Jamie</dd>
    <dd>Charlie</dd>
    <dt>Children</dt>
    <dd>Landry</dd>
    <dd>Oakley</dd>
    <dd>Skyler</dd>
    <dt>Pets</dt>
    <dd>Cat</dd>
    <dd>Dog</dd>
    <dd>Gerbil</dd>
</dl>
```

When that list is rendered, it will be displayed in such a way that the relationships between the terms (`dt`) and values (`dd`) are clear.

Parents
Jamie

Charlie

Children
Landry

Oakley

Skyler

Pets

Cat

Dog

Gerbil

Nested Lists

A nested list is a list within a list. If you've ever created a bulleted outline in a word processing document you probably used a variety of indentations and bullet point types to denote items that were subpoints of another item in the outline. This is the effect we're going for when we create nested lists.

To create a nested list, simply add a new list within a parent list like this:

```
<ul>
    <li>Item A</li>
    <li>Item B
        <ul>
            <li>Subitem B.1</li>
            <li>Subitem B.2</li>
        </ul></li>
    <li>Item C</li>
</ul>
```

When that list is loaded in the browser, the nested list will be indented further than the parent list, and a different type of item marker will be displayed.

- Item A
- Item B
  - Subitem B.1
  - Subitem B.2
- Item C

Nested lists aren't just used to organize the visual representation of information. Screen readers and other assistive technologies rely on the nested structure of complex lists to make sense of the hierarchy and logic of data within the list.

You could use assign classes to list items and use CSS to create the same visual effect as a nested list. However, if you did that, the hierarchical and logical structure of the list would be lost to website visitors using assistive technologies. In other words, don't use CSS to create nested lists visually, use HTML to create them.

Using Lists for Menus

One of the most common uses of lists is to create website navigation menus. Unordered lists are usually the list-of-choice for this purpose. With just a few lines of CSS we can convert an unordered list into an attractive horizontal navigation menu.

```
<style>
    #nav {
        background: lightgray;
        overflow: auto;
        }
    #nav li {
        float: left;
        list-style-type: none;
        padding: 10px;
        }
</style>
<ul id="nav">
    <li><a href="#Using_Lists_for_Menus">Home</a></li>
    <li><a href="#Using_Lists_for_Menus">About Us</a></li>
    <li><a href="#Using_Lists_for_Menus">Contact Us</a></li>
</ul>
```

If we load that code in the browser, you'll notice that each menu item changes when you hover over it.

Home About Us Contact Us

A lot more can be done with lists, CSS, and JavaScript to create interactive drop-down menus, and our menu tutorial will teach you how to create beautiful, modern, interactive, and well-organized menus.

Styling Lists

List typography is usually best styled to match the typography of the website's paragraph text. List-specific styling can be accomplished with CSS.

There are three list properties that can be styled with CSS:

- `list-style-type` : Defines the marker type that preceeds each list item. Common values include `disc` (the default unordered list style type), `decimal` (the default ordered list style type), circle, square, lower- or upper-roman, and lower- or upper-latin, although several additional styles may also be used.
- `list-style-position` : Determines whether the list item marker should be placed `inside` the content box, or `outside` of the content box in the item's left-hand padding area.
- `list-style-image` : An image can also be used as the item marker. This property is used to specify the image file to be used.

Each of these three properties can be applied seperately by using the individual property names or simultaneously with the `list-style` shorthand property. The `list-style` property syntax includes the list style type, position, and image values in that order. For example, if we wanted to select the square marker, position it inside the content box, and also specify an image file, our CSS would look something like this:

```
selector_for_the_target_list {
    list-style: square inside url("box_image.png");
}
```

Since we've specified both a list marker and an image, the image will be used unless it is unavailable, in which case the square marker will be displayed.

Closing Thoughts

Lists are one of the most diverse and effective tools in a web designers toolbox. They provide logical, hierarchical structure to data, and can be used in a variety of ways. Understanding the full breadth of what is possible with lists in HTML makes this powerful HTML element even more useful.

Jon Penland

Jon is a freelance writer, travel enthusiast, husband and father. He writes about web technologies such as WordPress, HTML, and CSS.

Related Elements

Element Name	Attributes	Notes
optgroup		The <optgroup> element is used to group together related <option> elements within a parent <select> drop-down list.
nav	gutter	The <nav> element identifies a group of navigation links. Links in a <nav> element may point to other webpages or to different sections of the same webpage.
HTML Tag	<ul type="">	The element is used to define an unordered list of items. Use an unordered list to contain elements that do not need to be presented in numerical order and can be rearranged without changing the meaning of the list.
li	value type	The element defines a list item that is part of an ordered and unordered list of items.
listing		The <listing> element was intended as a way to render HTML code on a page. It was never properly supported, and is now deprecated. Using <listing> will almost certainly result in unexpected results. Instead, use <code>, or place the content in a with the appropriate CSS styling.

ol	type start	The element is used to create an ordered list. An ordered list is created by nesting one or more elements between the opening and closing tags.
option	value selected	The <option> element is used in conjunction with the <select> element to create a drop-down menu in a web form. Each <option> element is displayed as an available option in the resulting drop-down menu.
dd		The <dd> element is used to pair a definition description with a sibling definition term enclosed in <dt> tags within a parent definition list.
dir		The <dir> element, deprecated beginning in HTML 4.01, was used to create a list of file names or the contents of a directory. An unordered list, created with the element, is the appropriate modern replacement for the <dir> element.
dl	compact	The <dl> element defines a description list.
dt		The <dt> element defines a term in a description list.

5 Hot Tips For Using Images In Your Site Builds Using HTML

Last Updated on July 22, 2019

Images are everywhere on the web. They are the most commonly shared content on social media and every modern web page contains at least one image while most contain a handful, a dozen, a hundred, or more. Yet despite their popularity, many websites are not using images to their full potential.

Contents [hide]

History of the `img` tag

Prior to 1993, the web was not a very friendly place for images. But in 1993 the `img` tag was proposed and Mosaic, the first browser to display images inline with text, was launched. Before Mosaic and the `img` tag images were loaded in a separate window or downloaded and opened with an image viewer. However, with Mosaic and the `img` tag added to HTML, the mid-1990s saw rapid growth in the popularity of the web. While the massive growth of the web in the early 1990s cannot be attributed entirely to improved support for images, it was one of a few factors that marked the transition of the web from being a network for researchers to a communications platform for the masses.

Five Tips for Using Images on the Web

Adding images to your website is great, but using images correctly is better. Images can improve the appearance of your site and can help explain concepts that are difficult to put into words. However, used improperly, images can be ineffective, create accessibility issues, and even land you in legal trouble. To help you avoid those pitfalls, here are our top five recommendations for using images on the web.

Only Use Images You Have the Right to Use

Have you ever heard someone say this before?

> *"Did you know you can use Google image search to find any image you want?"*

While that may technically be true, it's a terrible way to find images for your website. Many images are protected by copyright laws, meaning that using them without permission can land you in very hot water facing legal and financial penalties. This isn't something we're making up or exaggerating. Let the misfortune of teach you an important lesson: only use images you're sure you have the right to use. There are a few stock image sites that proudly proclaim that you can use their images free-of-charge without any attribution whatsoever. Stick to these sorts of sites for your images or buy images from stock photo sites. Wherever you source your images, be sure to follow licensing and attribution guidelines very carefully for every single image you post.

Keep Your Content and Website Accessible

If you are using an image that adds meaning or functionality to a web page, think about the experience of visitors using a text-only browser or screen reader. Will they be able to use the website and fully understand the web page content? Use the `alt` attribute to assign alternate text to your images to ensure your content and website are accessible.

Always Provide Alternate Text

As we just mentioned, alternate text is what website visitors rely on if they can't see your image. However, the use of `alt` text goes beyond just adding a short description to your images. Did you know there are times when you should use the `alt` attribute but leave it empty? It's true! If you have an image on your site that is purely aesthetic and does not contribute meaningful content to the site, use the empty attribute `alt=""` so that text-only browsers and screen readers will know to skip it entirely. The `alt` attribute should be used on every single `img` on your website. To learn more about this topic read our article on the rules of `alt` .

Know When to Use the Background-Image CSS Property

There are two ways to add images to a web page.

- With the `img` HTML element.
- With the `background-image` CSS property.

When should you use each? Use `img` when the image is part of the content of the web page and use `background-image` when the image is a part of the page presentation or visual design. While these guidelines cover the majority of images uses, if you want more specific guidelines you can find a great discussion on the topic at StackOverflow.

Optimize Images for the Web

Image files can be very large, and web pages containing multiple large images tend to load very slowly. Web pages that load slowly often have visitors who choose to click away from the site once they realize that it is loading slowly. Optimizing images for the web can reduce web page load times dramatically and give a boost to website traffic as fewer visitors are lost due to slow page loading. Take the time to learn about optimizing images for the web to ensure your website visitors have the best experience possible.

Two Other Ways to Add Images to a Web Page

HTML5 includes two new elements which can be used to add greater meaning to images and to give developers greater control over how visual elements are rendered different devices. These two elements are the `picture` and `figure` elements. These elements each have a specific purpose and aren't intended to be used as a general-purpose replacement for the `img` tag. Instead, think of these two elements are special-case elements you can use in unique situations to add greater meaning to certain kinds of images.

Picture Element

The `picture` element is used to identify the version of a picture that should be used based on the results of a media query. It is particularly useful when there are specific reasons why a certain version of a picture should be used rather than another for a specific screen size. If all you're trying to accomplish is to provide multiple image sizes for a variety of screen sizes and resolutions, the `img` tag and `srcset` attribute are a better option since they allow the browser to identify the best picture for the device. Use the picture element when the picture options available to the browser are not identical resized versions of each other and are different in some way. For example, if you have an image with a text overlay, when the image shrinks the text may need to increase in size relative to the image to remain readable on smaller screens. This is a great time to use the `picture` element so that you can explicitly tell the browser which image to use. On the other hand, if you just want to provide a smaller version of an image for use on smaller screens use the `img` tag and `srcset` attribute. Learn how to use `srcset` here. The `picture` element is not yet fully supported by all browsers. So if you use this new HTML5 element, be sure to provide an `img` fallback for visitors using unsupported browsers.

Figure Element

The `figure` element is an HTML5 feature used to identify an item on a web page that is relevant to the rest of the web page content, but does not depend on appearing in a specific position on the page. Content contained between `figure` tags should be able to be moved to a different position on the page without affecting it's meaning or the meaning of the rest of the page content. The way the HTML specification puts it is that "the figure element represents a unit of content…that is self-contained". Graphs, charts, and images are all common `figure` elements, but virtually any type of content, including videos, audio, and text can be nested between `figure` tags. The `figcaption` tag may optionally be added within the `figure` element to assign a caption to the content of the figure.

Conclusion

What would the web be without cat memes and pictures of your friends on social media? A much more boring place, that's for sure. Take the time to learn how to use images properly and you may find that your website's traffic rewards your efforts generously.

Jon Penland

Jon is a freelance writer, travel enthusiast, husband and father. He writes about web technologies such as WordPress, HTML, and CSS.

Related Elements

Element Name	Attributes	Notes
figure		The <figure> element identifies self-contained content related to the main content, such as an image, table, or chart. The <figcaption> element is often nested within a <figure> element to add a caption to the content identified by the <figure> tags.
figure		The <figcaption> element is used as a child of a parent <figure> element to attach a caption to the image, table, or

		chart contained in the <figure> element.
map	name	The <map> element is used in conjunction with one or more <area> elements to define hyperlinked regions of an image map.
img	crossorigin height srcset align alt border controls dynsrc hspace ismap longdesc loop lowsrc name naturalsizeflag nosave start suppress usemap width src	The <img> tag is used to insert an image into a document.
<area> HTML Tag	alt coords href nohref shape target title	The <area> element is used as a child of a <map> element to define clickable a region on an image map. Different regions of an image map can be hyperlinked to different locations by nesting multiple <area> elements in a single <map> element.

Tutorials and Resources

How to Create Image Transformations with JavaScript
How to Create Simple Image Transformations
Want To Create Images As Links With Or Without Borders? Here's How
4 Quick Steps To Make An Image Map In HTML (With Code Example)
The Rules of ALT

HTML Web Forms Tutorial For Coding Beginners

Last Updated on December 17, 2019

Web forms are used by virtually all websites for a wide range of purposes. Users of forums and social networks use forms to add content and interact with other users. Websites that can be customized to create a personalized experience, such as customizable newsfeeds, use forms to allow users to control the content that appears on the page. And nearly every website uses forms to allow website visitors to contact the organization or person administering the website. Web forms are made possible by the integration of multiple technologies:

- HTML to create the form fields and labels and accept user input.
- CSS to style the presentation of the form.
- JavaScript to validate form input and provide Ajax-enabled interactions.
- Server-side languages such as PHP to process form data.

In this guide, we're going to cover all of the HTML elements used to create web forms. We also have other tutorials that cover topics such as building a form, styling and designing forms, and ensuring form usability and accessibility.

Contents [hide]

Defining the Structure of a Form

Every web form must be wrapped in `<form>` tags. In most cases, all of the form fields will be nested between these tags. There are several attributes that may optional be used with the `form` element, including: **`accept-charset`** : This optional attribute is used to identify the character encodings acceptable to the server and code processing form input. If more than one encoding is specified, one space should be placed between each encoding. If left blank or not provided, the encoding will default to the same encoding as the document containing the form. **`action`** : This attribute is used to specify a URL where form data should be sent (for instance: http://example.com/form_file.php). This field was required prior to HTML5 but is now optional. **`autocomplete`** : Use this attribute if you want the visitors browser to suggest form responses based on saved entries. The default value is `autocomplete="on"` . If you turn autocomplete off you must also turn it off on every field that may allow autocompletion. **`enctype`** : Used to specify how form data should be encoded. Only used if the `method` attribute mentioned below is set to `post` . There are three possible values:

- `application/x-www-form-urlencoded` : The default value which replaces all spaces with "+" and converts all special characters to ASCII HEX values.
- `multipart/form-data` : Nothing is encoded. Input is uploaded to the server exactly as it is entered into the form.
- `text/plain` : Spaces are converted into "+" symbols, but not other characters are encoded.

`method` : Dictates the HTTP method a website visitors browser should use to submit form data. If `post` is specified, form data is enclosed in the body of the HTTP request. If `get` is specified, form data is appended to the end of the URL specified in the `action` attribute with a "?" symbol, data length is limited to 300 characters, and form input is visible and can be bookmarked. **`name`** : Similar to an `id` attribute, a `name` is a unique identifier that may only be used once within an HTML document and may be used to select the form with JavaScript or another scripting language. **`novalidate`** : Used to override the default validation of form data. **`target`** : Specifies where to display the response received after submitting the form.

- `_self` loads the response in the same frame.
- `_blank` opens a new window or tab.
- `_parent` is used when a form is nested in a descendant browsing context to load the response in the parent context and behaves the same as `_self` is there is no parent context.
- `_top` is similar to `_parent` but select the top level browsing context rather than the immediate parent context.

Grouping Form Fields

The `<fieldset>` element is used to group together related form fields. It may also be used to contain an entire form to provide a visual cue that distinguishes the form from surrounding content. The first element within a `fieldset` is usually a `<legend>` . The `legend` will be displayed a part of the `fieldset` border giving website visitors a clue about the purpose of a form. For example, if a contact form were nested within a web page full of otherwise unrelated content, the `fieldset` element could be used in combination with the `legend` element to isolate the contact form from the rest of the content both semantically and visually.

```
<p>Paragraph content.</p> <form>   <fieldset disabled>     <legend>Contact Form</legend>     <!--These element
s are discussed later in this tutorial-->     <p>Name: <input type="text" size="30"></p>     <p>Email: <input
type="text" size="30"></p>     <p>Subject: <input type="text" size="30"></p>     <p>Message: <textarea>Type yo
ur message here</textarea></p>     <p><input type="submit"></p>   </fieldset> </form>  <p>Additional paragraph
content</p>
```

When rendered in the browser, the contact form would be clearly separated from surrounding content.

Paragraph content.

Contact Form

Name:

Email:

Subject:

Message:

Type your message here

Additional paragraph content

The `fieldset` element is optional but is commonly used to group related elements on long and complex forms, or to isolate form fields from nearby elements when a form is presented along with non-form content. HTML5 added three attributes which can be applied to `fieldset` elements:

- `disabled` : Used to disable all of the forms in a fieldset. Note that we used this attribute in the example form above.
- `form` : Used to associate a `fieldset` with the `id` of one or more `form` elements. Note that browser support for this attribute is very limited.
- `name` : Associates a name with the `fieldset` .

The Input Element

Depending on the type of form you are creating, it's entirely possible to have a form that only includes two types of elements: one `form` element and one or more `input` elements. The `<input>` element is used to create a variety of different types of input fields for form users to interact with.

Common `<input type="">` Values

The `input` element is a very flexible element who appearance and behavior can change dramatically based on the `type` attribute applied to the element. The most common `type` values include: **`text`** : The default value for the `type=""` attribute. Defines a single line of text 20 characters wide. The width can be adjusted with the size attribute as you can see in the form code in our previous example. **`password`** : The password type is basically the same as the text field with the exception that characters entered into a password field are masked. **`radio`** : Radio buttons can be used to provide multiple options of which only one may be chosen. **`checkbox`** : Checkboxes are similar to radio buttons, but more than one selection can be active at a time. This means multiple values can be submitted with a set of checkboxes while a set of radio buttons will only accept one value. **`submit`** : The submit type value creates a form submission button. When clicked, the form will take the action specified in the `action` attribute associated with the `form` element. Many forms make use of just one or two `input` types, and most simple forms are built using just the types listed above. However, there are many additional types you can use to accept form data that doesn't fit into any of the types listed above.

Less Common `<input type="">` Values

These `input` types are less common than those listed above but are supported by virtually all browsers and can be used to build many different types of form inputs. **`button`** : If you want to run a `script` when a button is clicked, the `button` input type can be used to create a button which can be associated with a variety of actions. **`hidden`** : This attribute type is usually used simultaneously with the `value` attribute, which we'll cover momentarily, to add a pre-defined value to every form submission. For example, if you have contact forms on five different pages you could add `<input type="hidden" value="page_name">` to each form to submit the name of the page where the contact form was submitted from. **`reset`** : This type is used to create a `reset` button that will return all form fields to their default state. **`file`** : If you want to allow form users to upload and submit files, `<input type="file">` will provide that capability. **`image`** : In the past, it was common to use an image as a submit button. While this type value is still valid, it is not used very frequently in modern web form design. Instead, use CSS to style the button.

New `<input type="">` Values Added by HTML5

Several additional `input` types are defined by the HTML5 specification. Many of these types have limited browser support. So if you use them, be sure to check caniuse.com for browser support and provide adequate fallback options where appropriate. **`search`** : This is the proper `type` to specify if your form provides search functionality. Unlike most of the other types of inputs added in HTML5, `type="search"` is supported by all browsers. Just remember that this feature doesn't actually provide search functionality, it just creates a form input field designed to be used as a part of a search feature. **`color`** : When this input type is specified it will display a colored-button that launches a basic color selector tool. Set the default color value by using the `value` attribute and a hex color code like this: `<input type="color" value="#0000FF">` . **`number`** : This type produces a field for entering a number which has increment buttons on the righthand side of box. Limits can be placed on the range of acceptable entries with the `max` , `min` , and `step` attributes, but browsers that lack support for this element typically fallback to a standard text input that does not recognize these limits. **`range`** : Browser support for this element is pretty good with a few exceptions. Use this attribute to produce a slider which can be used to select a value within a specified range. For example, this code would produce a slider to select a number between 100 and 1000 in increments of 50: `<input type="range" name="range" min="100" max="1000" step="50">` . You'll need to pair the `range` element with another technique to provide a live preview of the selected value. Thankfully, HTML5 Doctor has a simple way to do this using the `output` element. **Date and Time Types**: HTML5 added a number of input types that can be used to specify time and date values. For example, the `date` type defines a control to enter a year, month, and day. To add time to the date input, use `datetime-local` . If you want time without date information use the `time` input type. Less specific input types include the `month` and `week` options which can be used to select a week or month within a year without specifying the day or time. Browsers have been slow to add support for this type, so be sure to check for browser support and provide fallbacks if you use this type of input. **Contact Detail Types**: Broad support is available for the three input types used to collect pertinent details like email addresses (`email`), telephone numbers (`tel`), and website URLs (`url`).

Common Input Attributes

While the `type` attribute is by far the most-used and most useful `input` attribute, there are several other attributes you will need to know to build useful forms. **`name`** : The `name` assigned to an `input` element will be submitted along with the value entered into the associated field. In other words, if the value "Fred" were entered into an `input` element with this code `<input type="text" name="first_name">` the value submitted would be "first_name=Fred". **`value`** : The value of an input element performs a different function depending on the type of input it is applied to. When applied to the `submit` , `reset` , or `button` types the value is used as the text on the button. When applied to text fields it provides the default value associated with the field. When associated with a checkbox or radio button, the value provides the value that will be associated with that field if selected.

```
<!--Will produce a button with the label "Send Now"--> <input type="submit" value="Send Now"> <!--Will produce a text field with "Google" preloaded in the field--> <!--Use when you know the value that should be submitted, not as placeholder--> <p>Who referred you to our website?:<br> <input type="text" name="Referrer" value="Google"></p> <!--If a radio button is selected the value "color=pink" or "color=red" will be submitted--> <p>Pink: <input type="radio" name="color" value="pink"></p> <p>Red: <input type="radio" name="color" value="red"></p>
```

`readonly` : When `readonly` is applied as an attribute of an `input` element the value in the field cannot be changed. JavaScript can be used to remove the `readonly` attribute after some other action is taken, such as clicking a button or selecting a checkbox. For example, `readonly` could be applied to a `submit` input type and removed when a checkbox was selected confirming that the user accepted the website's terms of service. **`disabled`** : We used this attribute with example form embedded earlier in this tutorial. Use this attribute to disable an entire form, fieldset, a single field. **`size`** : Use the `size` attribute with text `input` types to specify the visible width of the field without limiting the number of characters that may be entered into the field. **`maxlength`** : If you don't want to accept more than a certain number of characters in a given field, use `maxlength` to limit those fields to a defined number of characters. **`checked`** : If you want a checkbox or radio button to be preselected when a form loads apply this attribute to that `input` element. These attributes have broad support and are used commonly with forms you encounter every day.

New Attributes Added by HTML5

HTML5 added many new attributes which can be associated with `input` elements. Browser support for some of these elements is limited, so be sure to check for support and provide fallback options for users of unsupported browsers. **`autocomplete`** : If your form includes common fields, leaving autocomplete on will allow the visitors browser to suggest entries based on previously completed forms. **`autofocus`** : Use this

attribute to identify the form field that should be in focus when the form loads. **`multiple`** : The `multiple` attribute can be used with `email` and `file` input types to allow form users to input more than one value. When used for `email` inputs, more than one email address can be submitted by separating each address from the next with a comma. When used for `file` inputs, multiple files may be selected and submitted simultaneously. **`pattern`** : There may be times when you want to specify that the value of an `input` must meet certain criteria. For example, you may want to require that a password field includes at least one uppercase letter, one lowercase letter, one number, and meets a minimum length requirement. The `pattern` attribute can be used to create expressions against which `input` values are validated. Writing these expressions, referred to as Regular Expressions or RegExp, is beyond the scope of this tutorial. You can learn about regular expressions at Wikipedia and then write and test your expressions for free at RegExr. **`placeholder`** : Most forms include placeholder text which disappears as soon as you click into the field or begin typing. Use this attribute to add and define placeholder text for any inputs that accept text. **`required`** : If certain fields in a form are required, use this attribute to prevent submission of incomplete forms. **`form`** : If you ever need to place an `input` element outside of the `form` element, you can associate the displaced element by using the `form` attribute and applying a value that matches the `id` attribute applied to the form. **Modify Form Submit Button Behavior** There are five attributes that can be applied to `submit` and `image` input types to override the attributes applied to the parent `form` element. These attributes include:

- **`formaction`** : Define a different URL from the one identified in the parent form's `action` attribute to process a form submission. Often used when forms may be processed in more than one way to provide a variety of form submission options.
- **`formenctype`** : Specify an encoding type that should be used for form submissions. The value used overrides the value applied to the `enctype` attribute of the parent `form` element.
- **`formmethod`** : This attribute is used specify either the `get` or `post` method value and will override the `method` attribute applied to the parent `form` .
- **`formnovalidate`** : If you don't want form input to be validated when submitted you can use this attribute.
- **`formtarget`** : Override the `target` attribute applied to the parent `form` element by applying this attribute to a `submit` or `image` input type field.

Define the Size of `type="image"` : If you use the `image` input type to create a form submission button, you can control the size of the image using the `height` and `width` attributes. Alternatively, you can do the same thing with CSS. Most developers and designers would recommend avoiding these attributes and controlling button appearance with CSS. **Set Limits and Increments for Numeric Values**: You can use the `min` , `max` , and `step` attributes to define minimum and maximum values as well as acceptable increments falling between these values for any `input` that accepts numeric values.

Drop-Downs, Text Areas, & Buttons

Inputs aren't the only elements that can be used to create form fields. Other types of elements can be associated with forms to create drop-down lists or options, free-form text areas, and flexible buttons.

Pre-Populated Drop-Down Lists

To create a drop-down list of pre-populated options from which a website visitor can select an option, use the `select` element to create the field, and nest multiple `option` elements to create the various options that should appear in the drop-down menu. For example, to create a drop-down menu of pretentious color options for a fictional e-commerce store, the following code could be used:

```
<select name="color">   <option value="tan">Windswept Sand Dune</option>   <option value="green">Lush Forest</
option>   <option value="blue">Turbulent Waters</option>   <option value="black">Deep Night</option> </select>
```

Note that the `value` is what will actually be submitted with the form while the text between the opening and closing tags is what is presented to the visitor completing the form. For example, if a visitor selects "Lush Forest" the actual value submitted with the form will be `green` . Here's how our drop-down list shows up in the browser:

Windswept Sand Dune Lush Forest Turbulent Waters Deep Night

Free Form Text Areas

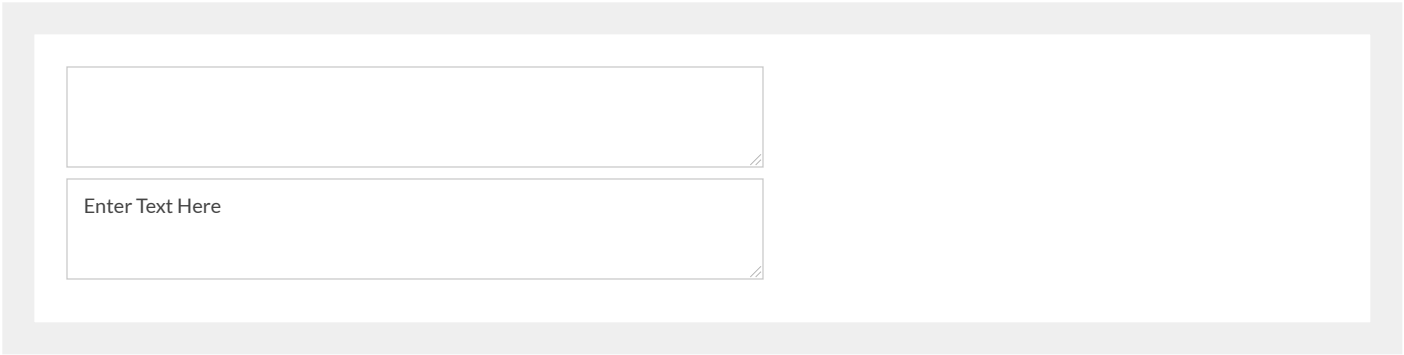
All of the text inputs we've seen so far, such as `<input type="text">`, only accept a single line of text. However, if you want to create a larger text area for longer text input a single line input field isn't going to work. The `textarea` element is the correct choice for creating a large text input area capable of accepting text input that won't render well on a single line. There are three parts to a `textarea`:

1. The `textarea` is created by inserting opening and closing tags. Any text nested between the tags will be loaded in the text area when the form loads and will be submitted along with the form unless the visitor submitting the form deletes the text out of the `textarea`.
2. If you want to define the size of the text area use the `rows` attribute to define the number of rows of text that should able to be displayed without resizing the text area.
3. To set a predefined width use the `cols` attribute. The value applied will be the number of characters that will appear on a single row before wrapping to the second row.

Text area elements are resizable. However, the `rows` and `cols` attributes will define the size of the `textarea` when it is first rendered by the browser and will also set the minimum space the area may be resized to fit.

```
<textarea rows="3" cols="60" placeholder="Enter Text Here"> </textarea> <br> <textarea rows="3" cols="60">   E
nter Text Here </textarea>
```

This code will produce two identically sized text areas that are three rows tall and can accept 60 characters per row. They will be resizable to any size larger than the default size. Note how the placeholder text was added to the first with the `placeholder` element but simply nested between the opening and closing tags in the second example. Below you can see how these two bits of code are rendered.



While `textarea` size can be specified using 'rows' and 'cols', it is a better practice to use CSS to style and size text areas. Many of the attributes that can be applied to `input` elements can also be applied to `textarea` elements. In addition, to those included in our example above, attributes that can be applied to text areas include: `autofocus`, `disabled`, `form`, `maxlength`, `name`, `readonly`, `required`, and `wrap`.

Flexible Buttons

There are two ways to create buttons for forms:

- `<input type="button/submit/reset/image">`
- `<button type="button/submit/reset/"></button>`

We've already talked about the `input` element and the different types that can be used to create buttons. So why is there another button? There are least two ways that `button` elements are different from their `input` cousins.

1. Because buttons are made with an opening and closing tag, different types of content – usually text and images – can be nested between the opening and closing tags and will be rendered on the button.
2. Buttons do not have to be associated with a `form` element. They can be used as standalone buttons to initiate scripts, as the content of an anchor element, and to perform other actions.

Form Elements Added in HTML5

Three new `form` elements were added in HTML5: **`datalist`** : Use this element to provide a list of suggested autocompletion values for an input element. In order to use the `datalist` element, first create an `input` element with a `list` attribute. Then create a `datalist` element with an `id` attribute. The value applied to the input list attribute must match the datalist id. Values are added to the `datalist` by adding `option` elements between the opening and closing `datalist` tags. Here's an example of how this all works:

```
<p>What is your favorite web technology?</p> <input type="text" list="web_technologies" name="web_technology">
<datalist id="web_technologies">   <option value="HTML">   <option value="CSS">   <option value="JavaScript">
<option value="jQuery">   <option value="PHP">   <option value="Bootstrap"> </datalist>
```

When we render that code in the browser and input element will appear. If we begin typing, autocomplete suggestions will be made based on the options included in the datalist. Note that users submitting the form aren't limited to selecting from one of these options. They can still choose to type a value that is not an included `option` if they wish to do so.

What is your favorite web technology?

`output` : Use this element to display the result of a calculation or user input. Associate it with an `input` element by using the `for` attribute with a value that matches the `id` of the relevant `input` element, or associate it with a `form` by adding a `form` attribute using a value that matches the `id` of the relevant `form` . The `output` element can also be paired with the `range` element to let form users know the exact value represented by the current position of the slider of a `range` element. Using the `output` and `range` elements in this way is beyond the scope of this introductory tutorial, but if you want to use these two elements together you can learn more about this technique at the HTML5 Doctor.

Next Steps

This tutorial has provided an overview of the elements used to build forms for the web. The next step is to try out some of the knowledge you've gained. Other tutorials in this section will walk you through the process of creating a reservation form, styling and designing forms, and ensuring that your forms meet usability and accessibility guidelines.

Frequently Asked Questions

How do you restrict a form field to only accept numbers?

In the past, restricting a field to numbers only required the use of JavaScript. However, with the release of HTML5, it's now simple to limit a field to numeric input only. Just apply the `number` value to the `type` attribute of the applicable `input` element. For example:

```
<input type="number">
```

When rendered, produces a text input field that will only accept numbers.

How do you restrict a form field to only accept alphanumeric characters?

In the past, the only way to limit field input to alphanumeric characters was to use jQuery or JavaScript and craft a custom function. However, HTML5 added the `pattern` attribute for `input` elements which can be used to restrict a form field to accept alphanumeric input only. Here's the code that will do the trick:

```
<form>  <input type="text" pattern="[a-zA-Z0-9]{5,8}" title="Type 5 to 8 alphanumeric characters">  <input t
ype="submit"> </form>
```

In this case, the `pattern` element will accept lowercase letters, uppercase letters, and numbers. The second part of the value in curly braces stipulates how many total characters may be entered into the field. Let's see how that looks in the browser.

The way it works is that when you attempt to submit values that don't meet the specified pattern the `title` attribute is displayed. So you should put some instructions inside the `pattern` attribute so that users can figure out what they've done wrong. Browser support for this relatively new attribute is really pretty good. IE 9 and earlier versions of IE don't support it and Opera Mini also lacks support. However, all other browsers do support the attribute.

How do you make a form submit when the user presses enter?

If you've come across a form that does not submit when you press enter, then someone has intentionally designed the form to behave that way – and they really shouldn't have done that. Web accessibility specs are clear: design forms to allow implicit submission. What is implicit submission? Submitting a form by pressing enter. Browsers are designed to support implicit submission. Here's how it works. Take this form for instance:

```
<form>  <label for="name">Name: </label><input type="text" name="name"><br>  <label for="age">Age: <input ty
pe="text" name="age"><br>  <input type="submit"> </form>
```

If you were focused on any element in that form and pressed enter, the form would be submitted. This is implicit submission and all modern browsers support this behavior. Use the `button` element in the form and you don't even have to use the `submit` value for the `type` attribute. Hit enter while focused on any element in this form and the form will still be submitted.

```
<form>  <label for="name">Name: </label><input type="text" name="name"><br>  <label for="age">Age: <input ty
pe="text" name="age"><br>  <button>Submit</button> </form>
```

So how do developers break this behavior? One way to get around this behavior–and to be clear, we don't recommend this–is to drop the `form` elements. The browser knows what to submit by grouping together everything between the `form` tags. Drop those tags and the browser doesn't know what to submit. Another way some developers manipulate browser behavior is to use CSS to make buttons rather than proper HTML elements, like this.

```
<style> .submitButton {  padding: 10px 20px;  margin-top: 10px;  background-color: #ddd;  border-radius: 5
px;  display: inline-block; } .submitButton:hover {  background-color: #ccc; } .submitButton:active {  back
ground-color: #ddd; } </style> <form>  <label for="name">Name: </label><input type="text" name="name"><br>
<label for="age">Age: <input type="text" name="age"><br>  <span class="submitButton">Submit</span> </form>
```

Which, when rendered by the browser, would produce a button that looked like a button but did not do anything when enter is pressed.

.submitButton{padding: 10px 20px; margin-top: 10px; background-color: #ddd; border-radius: 5px; display: inline-block;}.submitButton:hover{background-color: #ccc;}.submitButton:active{background-color: #ddd;} Name:

Age:
Submit

Usually, developers have good intentions when they do things like this. Typically, what they're trying to do is tie form validation to an `onClick` JavaScript event. Form validation is a good thing, but creating a barrier to accessibility in the name of form validation is not a good thing. So, what *should* you do instead of disabling implicit submission? Leave implicit submission intact and use JavaScript to add an event listener to each field. Use the event listener to trigger form validation like this.

```
document.getElementById('name').addEventListener('keypress', function(event) {   if (event.keyCode == 13) {
/* Run Form Validation JavaScript */   } };
```

That code leaves implicit submission intact, but still runs validation code when the user presses enter.

Jon Penland

Jon is a freelance writer, travel enthusiast, husband and father. He writes about web technologies such as WordPress, HTML, and CSS.

Related Elements

Element Name	Attributes	Notes
datalist		The <datalist> element is used to define autocompletion values for an associated <input> element. Suggested autocompletion values are added to a datalist by nesting one or more <option> elements between the opening and closing <datalist> tags.
input	step required readonly placeholder pattern multiple min max list height formtarget formmethod formenctype formaction form autofocus	The <input> element is used to create form fields that accept user input. Form <input> elements can be presented many different ways, including simple text fields, buttons, checkboxes, drop-down menus, and more, by setting the type attribute of the input element to the appropriate value.

	accesskey autocomplete border checked disabled maxlength language name size src type value	
label	for	The <label> element is used to associate a text label with a form <input> field. The label is used to tell users the value that should be entered in the associated input field.
legend	align	The <legend> element is used to add a caption to a group of related form <input> elements that have been grouped together into a <fieldset>.
select	disabled language multiple name onChange onFocus readonly size tabindex	The <select> element, used along with one or more <option> elements, creates a drop-down list of options for a web form. The <select> element creates the list and each <option> element is displayed as an available option in the list.
button	<button accesskey=""> disabled name type value onClick tabindex	The <button> element is used to create an HTML button. Any text appearing between the opening and closing tags will appear as text on the button. No action takes place by default when a button is clicked. Actions must be added to buttons using JavaScript or by associating the button with a form.
fieldset		The <fieldset> element may be optionally used to group together related fields in an HTML form.

form	<form target=""> action enctype method onSubmit onReset name	The <form> element is used to create an HTML form. The <form> element does not actually create form fields, but is used as a parent container to hold form fields such as <input> and <textarea> elements.

Tutorials and Resources

Getting Started With Forms
Form Styling and Design
Form Usability and Accessibility

HTML5 Basics For Everyone Tired Of Reading About Deprecated Code

Last Updated on January 15, 2020

You've probably seen many references to HTML5, along with mentions of certain features being "Deprecated" or "New" in HTML5. This can sometimes be confusing, as you find out that the way you thought you were "supposed" to do something is now *not the right way at all*.

It can also be annoying — especially since most of these "deprecated features" still work on most browsers. Why bother learning a new way of doing something if the old way works just fine?

And, of course, it can be difficult to find the right information sometimes. Content on the internet doesn't live forever… but almost. And the older content is, the more likely it is that you'll find it on a search engine. (All things being equal, Google and the others prefer older content. Also, older content has had more time to acquire backlinks.) This means that you will often find out-of-date tutorials when you are looking for information about HTML.

This article should clear up some of that confusion and get you on the right track with modern web development.

Contents [hide]

What is HTML?

(Skip ahead to What is HTML5 if you are already familiar with HTML generally.)

You probably already know that *HTML*, or HyperText Markup Language, is the language used for web documents. It is not a programming language, but rather a language that identifies the meaning, purpose, and structure of text within a document.

For example, consider this document:

All About Rats

Why Do Rats Make Great Pets?

Forget what you've learned from horror films, rats make great pets — especially for children. And unlike the popular hamster, they almost never bite.

You can easily understand the structure of the document because it is simple, and you are an intelligent human being who has read many documents in your life. But your web browser needs to understand the document in order to present it to you in an intelligent and meaningful way. So we use HTML tags (also called "elements") to identify the different pieces of the document.

For this simple document, we can add an `<h1>` tag to identify the main heading for the page (the title of the document), and `</h1>` to close it. Use the `<h2>` tag to identify the first headline within the content, and a `</h2>` tag to identify the block of content following it as a paragraph.

```
<h1>All About Rats</h1>
<h2>Why Do Rats Make Great Pets?</h2>
Forget what you've learned from horror films, rats make great pets -- especially for children. And unlike the
popular hamster, they almost never bite.
```

Now, a web browser can show you this document in a more meaningful way. This might seem trivial for a short document with one headline a single paragraph, but it can get complicated very fast. You've seen a lot of web pages, so you know that there are all sorts of things (not just headlines and paragraphs) that people need to represent on their web pages. Things like:

- Forms
- Lists
- Links to other pages
- Media

And that's just for starters. You also need to be able to identify sections of your document and provide metadata about the document itself. (*Metadata* is information about the document, such as the title, author, relevant keywords, and relationship to other documents.)

HTML History and Development

HTML has been around for a long time. Its roots go back to at least 1980, with Tim Berners-Lee's project ENQUIRE. And actually, the concept of hypertext goes back even further than that. The concept first appeared in the early 1940s, and was named and demonstrated in the 1960s.

In 1989, Lee proposed a new hypertext system based on the ideas of ENQUIRE (and other systems, such as Apple's HyperCard). This became the first version of what we now call HTML.

Since then, the language has been in constant development. The specification is managed by the World Wide Web Consortium (Berners-Lee is still the director, as of 2018), and the Web Hypertext Application Technology Working Group. (So, if you don't like HTML5, these are the people to blame.)

The language has evolved over all this time because web development has changed. We do things with web pages and HTML today that were never dreamt of by the early developers and implementers of the language. A web page is no longer just a document; it is likely to be a full-scale web application. And even when it is "just a document," we want search engines and other tools to understand the content of the website. We aren't just creating pages for human readers anymore, but for artificially-intelligent systems that collect and manipulate information.

Why did HTML have to change? Because the web has changed.

What is HTML5?

HTML5 is the latest specification of the HTML language, and represented a major break with previous markup practices. The purpose of the profound changes to the language was to standardize the many new ways in which developers were using it, as well as to encourage a single set of best practices with regards to web development.

Most of the individual changes are a result of larger objectives in the design of the language. These objectives primarily include:

- Encouraging semantic (meaningful) markup
- Separating design from content
- Promoting accessibility and design responsiveness
- Reducing the overlap between HTML, CSS, and JavaScript
- Supporting rich media experiences while eliminating the need for plugins such as Flash or Java

Getting a handle on HTML5 isn't just about learning which CSS features replace old HTML features. If you want to get an intuitive sense of HTML5, it is best to understand how these objectives affected the development of the language.

Encouraging Semantic Markup

Semantic markup means markup which has *meaning*, rather than markup which simply looks a certain way. For example, the `<h1>` tag implies that the content of the element is the title or headline of the entire document. That semantic meaning would be lost if we just made the text bold and large without using the appropriate tag.

HTML has always had a little bit of semantic markup available to it: heading tags, the link rel attribute, and document metadata. But it wasn't enough.

In previous versions of the language, common structural elements like page headers, navigation menus, and main content sections were all indicated with the same HTML element, the `<div>` tag . In HTML, there are a host of new semantic elements intended to indicate the basic structure of a page:

- `<header>`
- `<nav>`
- `<main>`
- `<article>`
- `<aside>`
- `<section>`
- `<footer>`

New text-level (inline) elements have also been introduced, such as `<address>` and `<time>` . These help search engines and other services to easily find information on a page, for display in other contexts. At the same time, existing inline elements which produce various effects like **bold**, *italic*, and underline have been refined or redefined to imply specific semantic meaning.

Separating Design From Content

Along with strongly encouraging semantic (meaningful) markup, the HTML5 specification strongly **discourages** non-meaningful markup — markup intended only to tell the browser how to display things. This includes things like:

- declaring fonts and text colors
- setting text alignment or justification
- placing borders on tables
- defining how text wraps around images

Most of the HTML features that allowed for these sorts of things have been completely deprecated. The few that are still officially supported come with warnings that they are usually not recommended practices.

There are primarily two reasons to prefer this separation:

- It is easier to maintain and redesign a site if the style declarations are confined to CSS
- Users view web content in a lot of different contexts — desktops, laptops, tablets, mobile phones, RSS readers, and many others. Styles and design decisions that make sense in one environment don't always make sense in another. So it is much better to provide semantic information and let the content be adapted to the context.

This last point is closely tied to…

Promoting Accessibility and Design Responsiveness

Not everyone interacts with the web the same way you do.

"Conventional" devices — desktops, laptops, tablets, and phones — present a wide range of screen sizes, screen aspect ratios, display resolutions, and user interaction experiences. This variety alone should be enough to encourage semantic and responsive design practices. But not everyone uses a "conventional" browser.

Blind and visually impaired persons browse the web also, and they use a variety of assistive technologies to do so. Screen readers that translate a site's content into speech, specialized browsers that strip out styling and present highly magnified or high-contrast text, braille interpreters, and keyboard-based navigation all allow those with non-standard vision to interact with websites.

And all of these technologies are hindered by markup which tries to "hard-code" design and styling into the content of a page.

Reducing the Overlap Between HTML, CSS, and JavaScript

Three languages define front-end web development — HTML, CSS, and JavaScript.

No one sat down at the beginning of the internet and figured what types of things belong to each language. They each evolved in parallel to each other, often overlapping in functionality and scope.

Besides the practical considerations enumerated above, there has also been a focus on defining the nature and purpose of these languages, and limiting them (or expanding them) so that they do what is in their nature to do:

- HTML — Content
- CSS — Design
- JS — Interactivity

Remembering this can help one determine which language to use, especially in cases where it is possible to do something in more than one way. For example, if you want to change the color of something, your very first thought should be to use CSS. On the other hand, if you want to change the color of something **in response to a user input**, you probably want to use JavaScript.

Supporting Rich Media Experiences While Eliminating the Need for Plugins Such as Flash or Java

As bandwidth and internet speed have increased, we have moved more and more toward using the internet as a media platform. HTML was originally created for (hyper-)text documents, with perhaps a few images, not rich media pages with audio and video.

When people first started adding these types of experiences to web pages, they required users to add special plugins to their browsers. These performed poorly, limited user options, and opened up security holes. They required developers to write core web page functionality in other languages like Flash or Java. The content was hidden from search engines and screen readers.

It was a mess.

Now, HTML5 provides support for media with elements like `<video>` and `<audio>`, while `<canvas>` provides a defined space for JavaScript-created drawing and graphics. New form elements, along with better integration between HTML5, CSS, and JavaScript has made it possible to create full-scale web applications using the three languages that are native to the web browser, without plugins or add-ons.

Why Should I Use HTML5?

The most straight-forward answer to that question is simply that it is the current, "right" version of the language.

But some people seem unconvinced by this fact. Older markup practices still work in most browsers — if you type `<img align="right" />` onto your web page, the text will flow around the image just the way you'd expect. Why not just do that? It's easier!

There are a number of reasons to prefer HTML5, and to avoid using any of the deprecated features. Some are practical, while others are more philosophical. Some are altruistic, while others are selfish.

- Easier to write
- Easier to maintain
- Easier to redesign
- Better for Search Engine Optimization
- Better for the blind and visually impaired
- Better for content aggregators and feed readers
- Better for users on mobile devices
- Better for users on slower internet connections
- Fewer chances of design breaks
- Easier to add media
- Easier to create interactive applications
- Deprecated features will likely stop being supported at some point, breaking your page

How To Use HTML5

You probably already know how to create HTML5 documents. The basics of the language are the same. There's a just a few things that are good to keep in mind.

Avoid Deprecated Features

<!– Browse the list on this page to make sure that you know which HTML features are no longer supported in HTML5. If you click on the links to the individual pages, you can learn more about why each feature was deprecated and how to accomplish similar effects using modern, standard features of HTML5 and CSS. –>
Make sure that you know which HTML features are no longer supported in HTML5. If you research the deprecated tags, you can learn more about why each feature was deprecated and how to accomplish similar effects using modern, standard features of HTML5 and CSS.

You don't have to memorize the list, though. All you really have to remember is that if you want to affect the way something **looks** on a page, you probably shouldn't attempt to do what you want with HTML. Nearly all of the HTML features that affected style or design have been deprecated, and the few that are left are only recommended in particular cases.

Learn to Use the New Features

Sometimes, if you don't know that something is available, you don't know to look for it. For example, if you didn't already know about the `<video>` element, you might not know just how easy it is to embed video on a web page.

<!– So it's a good idea to spend some time browsing the New Features list so that you are aware of what's available. –>
So it's a good idea to spend some time browsing the new Features so that you are aware of what's available.

Get Comfortable With CSS

Many of the deprecated features were used to achieve design and styling effects. These are now properly the domain of CSS. If you want to be a modern web front-end developer, you'll spend some time getting good at using CSS.

Use the HTML5 `<!DOCTYPE>` Declaration

All HTML5 documents should begin with a tag that indicated the document is, in fact, supposed to be valid HTML5. That looks like:

```
<!DOCTYPE html>
```

This should be the very first thing in a document, before the `<html>` tag, and before any whitespace.

Don't Close Null Tags

It's a little minor point, but…

A "null" or "empty" element is an element that has no content. These include:

- `<img>`
- `<br>`
- `<hr>`

Surprised that an `<img>` element has no content? The image itself is an attribute of the tag, not the content.

In some previous versions, HTML (those based on the XML standard) required these elements to be closed with a slash.

```
<!-- Self-closing null elements -->


<img src="example.jpg" />
<br />
<hr />
```

This is no longer required.

```
<!-- The HTML5 way -->


<img src="example.jpg">
```

```
<br>
<hr>
```

Validate Your Pages

Finally, you should make it a habit to validate your HTML documents against the specification. This means using an automated tool to check whether the markup adheres to the standard or not.

The W3C provides an official Markup Validation Service, which allows you to quickly check your pages against the HTML5 specification (and older specs too, if you like).

```
<!–
Deprecated Features in HTML5

New Features in HTML5

–>
```

Adam Wood

Adam is a technical writer who specializes in developer documentation and tutorials.

HTML Document Structure Before And After HTML5 – Here's What Changed

Last Updated on December 12, 2019

If you want to write semantic markup – and believe us, you do want to write semantic markup – then you need to structure HTML documents properly. The `html`, `head`, and `body` elements have been part of the HTML specification since the mid 1990s, and up until a few years ago they were the primary elements used to give structure to HTML documents. However, the situation has changed dramatically in the last few years as HTML5 has added a slew of new tags that can be used to add rich semantic meaning to the structure of an HTML document.

Contents [hide]

HTML Document Structure Before HTML5

If you've been using HTML for any time at all you know that every bit of HTML needs to be wrapped in `html` tags. An opening `<html>` tag should appear first and a closing `</html>` tag should appear at the bottom of the document. Every other bit of HTML should appear between those two tags.

The `head` element is the first element to appear after the opening `html` tag. In the document `head` we place things like the page `title` and `meta` data, we add JavaScript to our page with the `script` tag, and we [`link`] to external stylesheets and other resources.

On most webpages the `head` element is a very busy place. For this reason, we've created a tutorial that explains the tags that typically appear in the `head` element and what these tags are used for.

All of the content that is visible on a web page is nested between opening and closing `body` tags. The body is the primary container of the content that makes up a web page.

Up until HTML5, that was pretty much it for basic HTML document structure. All of our code was dropped in between the `body` tags and styled with CSS. However, now that HTML5 has broad support among modern browsers, it's time to implement the new HTML5 tags that will give our HTML documents a much more meaningful structure.

New Semantic Tags Added by HTML5

In this brief tutorial we'll touch on all of the new tags added as part of HTML5 to define the structure and content of a web page. The elements we're going to cover in this guide include:

- `header`
- `main`
- `nav`
- `article`
- `section`
- `aside`
- `address`
- `footer`

Using these elements isn't as complicated as it might appear at first glance, and most are fairly self-explanatory. We'll make a quick pass over each new element, and then draw up an HTML template you can use these new tags to add rich semantic meaning to your markup.

`<header>`

The `header` element is used to contain the content that appears at the top of every page of your website: the logo, tagline, search prompt, and possibly a navigational menu. In most cases, the `header` element is best positioned as a direct descendant of the `body` element, but it's also ok to place it inside the `main` element if you prefer.

`<main>`

Use the `main` element between `header` and `footer` elements to contain the primary content of your web page. The `main` element cannot be a descendant of an `article`, `aside`, `header`, `footer`, or `nav` element. Instead, it should be a direct descendant of the `body` element. Think of it as the direct replacement for the `div id="main"` you've used in the past to wrap up your entire page contents.

It's also ok to use more than one `main` element on a webpage. For example, if your blog homepage includes your five most recent posts, it would be appropriate to wrap each post in it's own `main` element – or you could wrap each in `article` tags.

`<nav>`

Navigational menus are commonly placed at the top of a web page, in a sidebar, or in the page footer. Wherever you happen to place a navigational menu, wrap it in `nav` tags. Note that you don't need to use `nav` tags for every link, just for blocks of links that provide either sitewide navigation or navigation for a specific part of a website.

`<article>`

If your website includes blog posts, articles, or any other content that could just as well appear on another website as syndicated content, wrap that content in an `article` post. You can use an `article` element just about anywhere other than nested within an `address` element, but in most cases an `article` element will be a direct descendant of a `main` element or of a `section` element that is a direct descendant of a `main` element.

`<section>`

The `section` element is used to identify content that is a major sub-section of a larger whole. For example, if you've posted a long-form ebook in HTML format, it would be reasonable to wrap each chapter in a `section` element. Likewise, if you have a sidebar (semantically wrapped in `aside` tags) that contains four sections – ads, a search prompt, related posts, and a newsletter signup form – it would be ok to wrap each of these four sections in `section` tags since a written outline of the sidebar contents would include a line item for each of the four sections.

There is some confusion about when to use a `section` and when to use a `div`. Here's a good rule of thumb to help you know when to use each:

- Use a `div` if you're wrapping up some content purely to make it easier to style the content or to make it easier for some JavaScript to get ahold of it.
- Use a `section` if you would list the content as an item when writing out an outline of the document.

`<aside>`

If your website contains information that isn't directly related to the main content of the page, it would be appropriate to wrap that information in `aside` tags. For example, if you write a post that includes some technical terms, and you add definitions for those terms in a sidebar, it would make sense to wrap those definitions in `aside` tags. It is also common for the entire sidebar of a blog-type website to be wrapped in `aside` tags to make it clear that the sidebar is not part of the primary content of the page.

`<address>`

The `address` element provides contact information for the nearest parent `article` or `body` element that contains it. Use the `address` element inside an `article` to provide contact information for the article's author. Use it outside of an `article` in the `main` or `footer` elements, or as a direct descendant of the `body` element, to provide contact information for the website's owner.

`<footer>`

The `footer` appears at the bottom of a section of a document. Typically, the `footer` is a direct descendant of the `body` element, but it can also be used within a `main` element, a `section`, or an `article`. The most common use of the `footer` element is to place it at the bottom of an HTML document to contain things like a copyright notice, links to related content, `address` information about the owner of the website, and links to administrative things like privacy policies and website's terms of service.

You may also use the `footer` element within an `article` to provide metadata about that particular article. For example, if `article` tags have been used to wrap a forum post, it would be appropriate to wrap copyright information and the date and time the post was made in a `footer` element and place it at the bottom of the `article`.

An HTML Document Template

The template below will show you how all of these elements are properly nested together. We invite you to copy it and use it as a boilerplate template for all of your HTML documents.

```
<html>
  <!--Only the head and body elements are supposed to be direct descendants of the
  html element. All others should be descendants of either the head or body-->
  <head>
    <!--The head element must be a direct descendant of the html element-->
    <!--The head element is a very busy place for most websites, so we've created
    a tutorial to walk you through the different elements and tasks accomplished
    in the head element. You can find it at the following address:
    https://html.com/document/metadata/ -->
    <title>Your Webpage Title Goes Here</title>
  </head>
```

```
  <body>
    <!--The body element contains the full visible content of the web page-->
    <header>
      <!--The header typically includes your logo, tagline, and may contain a nav
      element-->
      <nav>
        <!--The nav element isn't used for every single link but for navigational
        menus-->
      </nav>
    </header>
    <main>
      <!--The main element cannot be used inside of anything other than the body
      element. It is intended to hold the main content of the page.-->
      <nav>
        <!--You can use a nav element just about anywhere-->
      </nav>
      <article>
        <!--If your web page contains a blog post or news article it makes sense
        to wrap the whole article in article tags-->
        <aside>
          <!--The aside tag can be used within an article or outside of it. It
          is used to mark content that is related but not central to the main
          content of the page-->
        </aside>
        <section>
          <!--Sections are used to seperate major parts of an element, such as
          chapters of an HTML ebook, or to cordone off the comments section
          from the rest of the main element-->
        </section>
        <address>
          <!--An address element inside of an article element is used to provide
          contact info for the author of the article-->
        </address>
      </article>
      <aside>
        <!--The aside element would also be used to mark a sidebar if used
        outside of the main element-->
        <section>
          <!--Within a sidebar you could use section elements to identify the
          different parts of the sidebar. For example, you could put adds in
          one section, related posts in a second section, and a newsletter
          signup form in a third section element.-->
        </section>
      </aside>
    </main>
    <footer>
      <!--The footer typically contains links to things like About Us, Privacy
      Policy, Contact Us and so forth. It may also contain a nav, address,
      section, or aside element.-->
      <address>
        <!--Put an address element in the footer and you're indicating that
        the contact info within the element is for the owner of the website
        rather than the author of the article.-->
      </address>
    </footer>
  </body>
</html>
```

Jon Penland

Jon is a freelance writer, travel enthusiast, husband and father. He writes about web technologies such as WordPress, HTML, and CSS.

Related Elements

Element Name	Attributes	Notes
main		The <main> element is used to denote the content of a webpage that relates to the central topic of that page or application. It should include content that is unique to that page and should not include content that is duplicated across multiple webpages, such as headers, footers, and primary navigation elements.
header		The <header> element is used to identify content that precedes the primary content of the web page and often contains website branding, navigation elements, search forms, and similar content that is duplicated across all or most pages of a website.
footer		The <footer> element is a structural element used to identify the footer of a page, document, article, or section. A <footer> typically contains copyright and authorship information or navigational elements pertaining to the contents of the parent element.
aside		The <aside> element is used to identify content that is related to the primary content of the webpage, but does not constitute the primary content of the page. Author information, related links, related content, and advertisements are exampes of content that may be found in an aside element.

article		The <article> element identifies a self-contained piece of content which could theoretically be distributed to other websites and platforms as a stand-alone unit. The <article> element is a good choice to contain entire blog posts, news articles, and similar content.
title		The <title> element is a required HTML element used to assign a title to an HTML document. Page titles are not displayed in the browser window, but they are used as the page name by search engines and displayed by browsers in the title bar, on the page tab, and as the page name of bookmarked webpages.
isindex	action prompt	The <isindex> element was used to create a single line search prompt for querying the contents of the document. Implementation of the element was inconsistent and the functionality duplicated by the <form> and <input> elements. As a result, <isindex> was deprecated in HTML 4.01.
meta	content http-equiv <meta name="">	The <meta> element is used to add machine-readable information to an HTML document. Information added with the <meta> tag is not displayed to website visitors but is provided for use by browsers and web crawlers.
html comment		This element is used to add a comment to an HTML document. An HTML comment begins with <code><!––</code> and the comment closes with <code>––></code>. HTML comments are visible to anyone that views the page source code, but are not rendered when the HTML document is rendered by a browser.
DOCTYPE		The <!DOCTYPE html> declaration is used to inform a website visitor's

		browser that the document being rendered is an HTML document. While not actually an HTML element itself, every HTML document should being with a DOCTYPE declaration to be compliant with HTML standards.
base	target href	The <base> element is used to identify a base URL upon which to build all relative URLs that appear on a webpage. In addition, if the <base> element has a target attribute, the target attribute will be used as the default attribute for all hyperlinks appearing in the document.
body	background bgcolor bgproperties stylesrc text scroll topmargin onUnload onLoad onFocus	The <body> element contains the entire content of a webpage. It must be the second element inside of the parent <html> element, following only the <head> element.
html		The <html> element is used as a container for all of the HTML of an entire document.
head		The <head> element contains information about an HTML document that is used by browsers and web crawlers but is not displayed to website visitors.
div	align	The element defines an arbitrary block of content which can be placed and styled as a single unit.

Tutorials and Resources

What Is Metadata In HTML Documents?: Head Elements Explained

What On Earth Is Semantic Markup? (And Why Should You Learn To Write It)

Last Updated on November 18, 2019

As you learn about HTML and the Web you may find that you encounter one specific word repeatedly that is often left undefined. That word is *semantic*.

You may read statements such as "we went looking for a semantic element" or "We try and be as semantic as we can", yet never get a clear picture of what the word *semantic* means.

In this article, we'll explore the world of semantic markup, come up with a working definition of the term, and apply the concept to the way we write HTML markup.

Contents [hide]

What is Semantic Markup?

According to Dictionary.com, *semantics* refers to the correct interpretation of the meaning of a word or sentence.

To use a word semantically is to use it in a way that is properly aligned with the meaning of the word. When we misuse a word we are not using it semantically.

Many HTML tags have semantic meaning. That is, the element itself conveys some information about the type of content contained between the opening and closing tags.

For example, when a browser encounters an `h1` heading it interprets that tag to mean that the contents of the `h1` element constitute the most important heading of the section that contains the element.

The semantic meaning of an `h1` tag is that it is used to identify the most important header of a specific web page or section.

Two Practices that Enable Semantic Markup

There are two different practices that must be put into place if we are going to write semantic markup.

1. Semantic markup requires that HTML elements be used according to their intended purpose.
2. Semantic markup requires the separation of content and presentation.

Using HTML Elements Correctly

When writing semantic markup, we use HTML tags to tell browsers something about the contents of the element. In semantic markup, tags are no longer just a way to get content to show up on a web page in a human-readable format.

The tags themselves become a way to tell a machine (whether a browser, a computer, a smartphone, or another smart device) something about the meaning of the content.

To write semantic markup, we must use HTML tags correctly so that our markup is both human-readable and machine-readable.

Seperating Content and Presentation

In the past, it was common to use markup to define styles and to control web page layout.

Heading levels were selected not based on hierarchy but based on the styles applied by the web browser, tables were used for web page layout rather than to organize tabular data, some HTML tags (such as `frameset`) were created for the express purpose of defining web page layout, and so forth.

When we write semantic markup we can no longer select HTML elements based on visual presentation. Instead, we select HTML elements based on their semantic meaning, and then use CSS to define the visual presentation of our content.

When writing semantic markup, the presentation of web page elements is kept completely separate and distinct from the markup of the content itself.

Defining Semantic Markup

With those two practices in mind, we can define semantic markup in this way:

Semantic markup is the use of a markup language such as HTML to convey information about the meaning of each element in a document through proper selection of markup elements, and to maintain complete separation between the markup and the visual presentation of the elements contained in the document.

Why is Semantic Markup Important?

Good CSS can make bad markup invisible to the average website visitor. However, no amount of styling will make bad markup more meaningful to a computerized visitor such as a search engine web crawler, browser translation tools, or assistive technologies such as screen readers.

According to Bruce Lawson, the semantic use of HTML elements “enhances accessibility, searchability, internationalization, and interoperability.” In other words, writing semantic markup is mandatory if you want your website to be accessible to all visitors, to achieve a high search engine ranking, to be available to visitors from around the world, and to interface effectively with other web services.

Writing semantic markup is about creating web content that is both human and computer readable. When the web can be read equally well by both humans and computers, it becomes more accessible since computers are better able to analyze its contents, index it, deliver it, and developers are better able to tie different sources of information together into new web services.

How Do We Write Semantic Markup?

We write semantic markup by selecting and using HTML tags properly, and by selecting tags that convey something about the information marked by the tags.

There are elements in HTML that are semantic and elements that are non-semantic. Examples of non-semantic elements are `div` and `span`. These tags don't tell the computer anything about the meaning of the contents of the element.

While useful, and fine to use in some cases, if a semantic tag is available and appropriate for a specific use, use it before resorting to a non-semantic tag.

Many semantic tags come from the analysis of web page markup completed by companies like Google and Opera. What these companies have found is that many websites use `id` and `class` attributes to hint at the meaning of the contents of non-semantic elements.

For example, they found lots of divs that looked like this: `<div id="nav">`, `<div id="header">`, and `<div id="footer">`. Findings like these helped the W3C identify and target new semantic tags to include in HTML5 such as: `nav`, `header`, `footer`, `article`, and `aside`. We can group the most common and important semantic elements into four categories:

- Document structure tags
- Textual meaning tags
- Media type tags
- Correlation tags

Document Structure

In the past, the `div` element was the main way sections of a website were identified and grouped. However, with the release of HTML5, we have several new tags to work with that provide semantic meaning in addition to the grouping attributes offered by the `div` tag:

- `header`: A container to be used for a web page header which typically contains the site logo, heading elements, and site navigation.
- `footer`: A container to be used for a web page footer which typically contains authorship, contact, and copyright information in addition to navigational links and a link back to the top of the web page.
- `main`: A high-level element used to contain all of the content that is unique to a single web page and not repeated across multiple web pages.
- `nav`: An element to contain blocks of site navigation links. This element is typically placed in the page `header` and `footer`, and may also be used in an `aside` (sidebar) element as well.
- `section`: The `section` element is used to mark off sections of a document, such as chapters or major sections of a long form post.
- `aside`: Use to identify content that is related to the main content on the page but not part of the primary flow of the document. For example, the `aside` element may contain a glossary definition of a term that appears in a blog post or it may contain advertisements related to the contents of the page.
- `article`: The article element is used to identify a block of content suitable for reuse and syndication in other settings, such as a blog post or technical article.

Review our Document Tutorial to learn more about using these semantic tags that add structure to a web page.

Textual Meaning

In the early days of the web it was common to see markup like this:

```
<style> .italics {     font-style: italic; } </style> <p>Some paragraph content including one <span class="ita
lics">italicized</span> word.</p>
```

Today we (hopefully) wouldn't dream of doing something like that since the `span` element tells the browser and other computerized visitors absolutely nothing about the meaning or purpose of the text nested in the between the opening and closing tags. Rather than use the non-semantic `span` tag, we'd add `em` tags around the words that should appear in italics. By using `em` tags, visitors using screen readers or other computers accessing the content would understand that the tags were applied to add emphasis to the tagged content. The `em` element is just one example of how HTML tags add semantic meaning to textual content. Other examples include:

- `h1`, `h2`, `h3`, `h4`, `h5`, and `h6`: Heading element tags are used to identify text that should appear as a heading. The highest level, or most important, heading is `h1` which is followed by heading levels `h2` through `h6` in order of descending importance.
- `strong`: Text that is marked with `strong` tags is given added importance and is usually displayed in a **bold** typeface.
- `mark`: The `mark` tag is used to highlight text of specific importance in a specific context. For example, it can be used to highlight every occurrence of a search term in a search results page.
- `cite`: The `cite` element is used to identify the original work from which a bit of content originates.
- `blockquote` and `q`: The `blockquote` and `q` (quote) elements are used to identify text that is a direct quotation from another source.
- `time`: The `time` element can be used to tell browsers, web crawlers, and other smart devices that a specific bit of content represents time on a 24-hour clock or a specific calendar date.

Our Fonts and Web Typography Tutorial provides a great deal more detail surrounding the proper use of these tags to assign semantic meaning to textual content.

Media Type

HTML5 also includes three tags that identify the type of media served up between the tags. These tags serve a dual purpose. First, they signal to the browser the need to queue up a specific technical resource such as a video playback engine. Second, they assign semantic meaning to the content.

- `audio`: Used to add one or more sources of audio content to a document and to allow the browser to pick the best option based on the visitor's device and browser.
- `video`: Similar to the `audio` element but used to add video content to a markup document.
- `picture`: The picture element is used to allow a web browser to pick the best image from the available options based on the results of a media query.

You can learn more about embedding `audio` and `video` elements in our HTML5 Media Tutorial. In addition, our article on the use of images on the web provides additional information on when to use the `picture` element and when to stick with the `img` element.

Correlation Tags

Several HTML elements are used to signal a correlation between multiple elements. For example, the use of an ordered list (`ol`) tells the browser that the items on the list are related to eachother and need to appear in a specific order. Other elements that are used to signal correlation between multiple elements include:

- `ul`: Unordered lists are used to signal a relationship between the items on the list and to indicate that they do not need to be understood in a specific order. Read our Lists Tutorial to learn more about how to use both ordered and unordered lists.
- `figure`: The `figure` element is used to group together a piece of content, such as an image, chart, graph, or text, and a caption marked off by `figcaption` tags. By nesting the caption and the content between `figure` tags a relationship between the nested elements is identified. Our images page contains more information about implementing this helpful tag.
- `address`: This attribute is used to associate contact information with the parent element that contains the `address` element. For example, when added to an `article`, the `address` element provides contact information for the article author, and when added to a web page `footer` the `address` identifies contact information for the web page owner.

Closing Thoughts

If you're new to HTML take the time to learn how to use all of these different HTML tags semantically. If you aren't sure that you're using the right tag, take a few minutes and do some research. As we've seen, using the right tag is important. If you've been working with HTML for a while now, take the time to learn about the new HTML5 elements and how to properly use them. HTML has grown increasingly complex over the last several years, and it can be tempting to keep using `div` elements with `class` and `id` attributes, but the accessibility and interoperability promise of semantic HTML5 tags is reason enough to embrace these new semantic elements. As Internet access becomes more widespread, smart devices proliferate, and the web further integrates into the fabric of modern society, the need for markup to be semantically accurate becomes increasingly evident. No longer is web page content isolated to desktop computers and accessed with just a few web browsers. Today, the semantic web is growing up all around us. By ensuring that every bit of markup you touch is semantic, you play a part in enabling the ongoing growth of the increasingly interconnected web.

Jon Penland

Jon is a freelance writer, travel enthusiast, husband and father. He writes about web technologies such as WordPress, HTML, and CSS.

Related Elements

Element Name	Attributes	Notes
section		The <section> element is a structural HTML element used to group together related elements. Each <section> typically includes one or more heading elements and additional elements presenting related content.
progress		The <progress> element is used to create a progress bar to serve as a visual demonstration of progress towards the completion of task or goal. The max and value attributes are used to define how much progress (value) has been made towards task completion (max).
output		The <output> element is used to display the result of a calculation. The <output> element is typically used in conjunction with a parent <form> and sibling <input> elements to perform a calculation. The actual calculation is typically completed using JavaScript.

menuitem		The <menuitem> element is used to add menu items and commands to contextual pop-up menus (the menus that appear when you right-click in a web browser).
main		The <main> element is used to denote the content of a webpage that relates to the central topic of that page or application. It should include content that is unique to that page and should not include content that is duplicated across multiple webpages, such as headers, footers, and primary navigation elements.
aside		The <aside> element is used to identify content that is related to the primary content of the webpage, but does not constitute the primary content of the page. Author information, related links, related content, and advertisements are exampes of content that may be found in an aside element.
article		The <article> element identifies a self-contained piece of content which could theoretically be distributed to other websites and platforms as a stand-alone unit. The <article> element is a good choice to contain entire blog posts, news articles, and similar content.
acronym		The <acronym> element and title attribute was used to associate a full-text explanation with an acronym. The <acronym> element has been deprecated in HTML5 and <abbr> should be used instead.
abbr	title	The <abbr> element is used along with a title attribute to associate a full-text explanation with an abbreviation or acronym. Website visitors do not see the text in the title attribute, but browsers, search

		engines, and assistive technologies do use this information.
menu		The <menu> element defines an instance of a menu. This experimental HTML feature has very limited browser support, but may soon be an effective way to add menu items to context menus and to create interactive web application menus.
Proper Use Of The Strong Element In HTML (Plus Code Example)		The element is used to identify text that is of greater importance than the surrounding text. By default, all browsers render text in a bold typeface.
address		The <address> element identifies contact information relevant to the current site, page, document, section, or article. It should not be used to identify addresses in any other context.
dfn		The <dfn> element is used to identify the defining instance of a term in an HTML document. When a term is wrapped in <dfn> tags, browsers and web crawlers will understand that nearby text contains a definition of the term.
headlines	align	The <h1>, <h2>, <h3>, <h4>, <h5>, and <h6> elements are used to create headings in descending order of importance where <h1> is the most important and <h6> the least.
ins		The <ins> element is used to identify text that has been inserted into a document. It is often paired with a <del> element which identifies deleted text replaced by the text contained in the <ins> element.