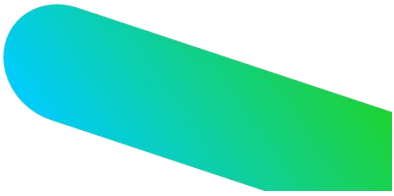
HTML5 & Css Hand out



What is New in HTML5?

The DOCTYPE declaration for HTML5 is very simple:

```
<!DOCTYPE html>
```

The character encoding (charset) declaration is also very simple:

```
<meta charset="UTF-8">
```

HTML5 Example:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Title of the document</title>
</head>

<body>
Content of the document......
</body>

</html>
```

The default character encoding in HTML5 is UTF-8.

New HTML5 Elements

The most interesting new HTML5 elements are:

New **semantic elements** like `<header>`, `<footer>`, `<article>`, and `<section>`.

New **attributes of form elements** like number, date, time, calendar, and range.

New **graphic elements**: `<svg>` and `<canvas>`.

New **multimedia elements**: `<audio>` and `<video>`.

New HTML5 APIs (Application Programming Interfaces)

The most interesting new APIs in HTML5 are:

- HTML Geolocation
- HTML Drag and Drop
- HTML Local Storage
- HTML Application Cache
- HTML Web Workers
- HTML SSE

Tip: HTML Local storage is a powerful replacement for cookies.

HTML History

Since the early days of the World Wide Web, there have been many versions of HTML:

Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+

1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition
2017	W3C Recommendation: HTML5.2

From 1991 to 1999, HTML developed from version 1 to version 4.

In year 2000, the World Wide Web Consortium (W3C) recommended XHTML 1.0. The XHTML syntax was strict, and the developers were forced to write valid and "well-formed" code.

In 2004, W3C's decided to close down the development of HTML, in favor of XHTML.

In 2004, WHATWG (Web Hypertext Application Technology Working Group) was formed. The WHATWG wanted to develop HTML, consistent with how the web was used, while being backward compatible with older versions of HTML.

In 2004 - 2006, the WHATWG gained support by the major browser vendors.

In 2006, W3C announced that they would support WHATWG.

In 2008, the first HTML5 public draft was released.

In 2012, WHATWG and W3C decided on a separation:

WHATWG wanted to develop HTML as a "Living Standard". A living standard is always updated and improved. New features can be added, but old functionality cannot be removed.

HTML5 Browser Support

HTML5 Browser Support

HTML5 is supported in all modern browsers.

In addition, all browsers, old and new, automatically handle unrecognized elements as inline elements.

Because of this, you can "teach" older browsers to handle "unknown" HTML elements.

You can even teach IE6 (Windows XP 2001) how to handle unknown HTML elements.

```
header, section, footer, aside, nav, main, article, figure {
  display: block;
}
```

Add New Elements to HTML

You can also add new elements to an HTML page with a browser trick.

This example adds a new element called `<myHero>` to an HTML page, and defines a style for it:

```
<!DOCTYPE html>
<html>
<head>
<script>document.createElement("myHero")</script>
<style>
myHero {
  display: block;
  background-color: #dddddd;
  padding: 50px;
  font-size: 30px;
}
</style>
</head>
<body>

<h1>A Heading</h1>
<myHero>My Hero Element</myHero>

</body>
</html>
```

The JavaScript statement `document.createElement("myHero")` is needed to create a new element in IE 9, and earlier.

Problem With Internet Explorer 8

You could use the solution described above for all new HTML5 elements.

However, **IE8 (and earlier) does not allow styling of unknown elements!**

Thankfully, Sjoerd Visscher created the HTML5Shiv! The HTML5Shiv is a JavaScript workaround to enable styling of HTML5 elements in versions of Internet Explorer prior to version 9.

You will require the HTML5Shiv to provide compatibility for IE Browsers older than IE 9.

Syntax For HTML5Shiv

The HTML5Shiv is placed within the `<head>` tag.

The HTML5Shiv is a javascript file that is referenced in a `<script>` tag.

You should use the HTML5Shiv when you are using the new HTML5 elements such as: `<article>`, `<section>`, `<aside>`, `<nav>`, `<footer>`.

You can download the latest version of HTML5shiv from github or reference the CDN version at https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js

Syntax

```
<head>
  <!--[if lt IE 9]>
    <script src="/js/html5shiv.js"></script>
  <![endif]-->
</head>
```

HTML5Shiv Example

If you do not want to download and store the HTML5Shiv on your site, you could reference the version found on the CDN site.

The HTML5Shiv script must be placed in the `<head>` element, after any stylesheets:

Example

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<!--[if lt IE 9]>
  <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
<![endif]-->
</head>
<body>

<section>

<h1>Famous Cities</h1>

<article>
<h2>London</h2>
<p>London is the capital city of England. It is the most populous city in
the United Kingdom, with a metropolitan area of over 13 million
inhabitants.</p>
</article>

<article>
<h2>Paris</h2>
<p>Paris is the capital and most populous city of France.</p>
</article>

<article>
<h2>Tokyo</h2>
<p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area,
and the most populous metropolitan area in the world.</p>
</article>

</section>

</body>
</html>
```

HTML5 New Elements

New Elements in HTML5

Below is a list of the new HTML5 elements, and a description of what they are used for.

New Semantic/Structural Elements

HTML5 offers new elements for better document structure:

Tag	Description
<article>	Defines an article in a document
<aside>	Defines content aside from the page content
<bdi>	Isolates a part of text that might be formatted in a different direction from other text outside it
<details>	Defines additional details that the user can view or hide
<dialog>	Defines a dialog box or window
<figcaption>	Defines a caption for a <figure> element

<figure>	Defines self-contained content
<footer>	Defines a footer for a document or section
<header>	Defines a header for a document or section
<main>	Defines the main content of a document
<mark>	Defines marked/highlighted text
<meter>	Defines a scalar measurement within a known range (a gauge)
<nav>	Defines navigation links
<progress>	Represents the progress of a task
<rp>	Defines what to show in browsers that do not support ruby annotations
<rt>	Defines an explanation/pronunciation of characters (for East Asian typography)

<ruby>	Defines a ruby annotation (for East Asian typography)
<section>	Defines a section in a document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time
<wbr>	Defines a possible line-break

New Input Types

New Input Types	New Input Attributes

- color
- date
- datetime
- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week

- autocomplete
- autofocus
- form
- formaction
- formenctype
- formmethod
- formnovalidate
- formtarget
- height and width
- list
- min and max
- multiple
- pattern (regexp)
- placeholder
- required
- step

HTML Basic Examples

HTML Documents

All HTML documents must start with a document type declaration: `<!DOCTYPE html>`.

The HTML document itself begins with `<html>` and ends with `</html>`.

The visible part of the HTML document is between `<body>` and `</body>`.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

HTML Headings

HTML headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading:

Example

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

HTML Paragraphs

HTML paragraphs are defined with the `<p>` tag:

Example

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

HTML Links

HTML links are defined with the `<a>` tag:

Example

```
<a href="https://www.w3schools.com">This is a link</a>
```

The link's destination is specified in the `href` attribute.

Attributes are used to provide additional information about HTML elements.

You will learn more about attributes in a later chapter.

HTML Images

HTML images are defined with the `<img>` tag.

The source file (`src`), alternative text (`alt`), `width`, and `height` are provided as attributes:

Example

```
<img src="hello.jpg" alt="W3Schools.com" width="104" height="142">
```

HTML Buttons

HTML buttons are defined with the `<button>` tag:

Example

```
<button>Click me</button>
```

HTML Lists

HTML lists are defined with the `<ul>` (unordered/bullet list) or the `<ol>` (ordered/numbered list) tag, followed by `<li>` tags (list items):

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>

<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

HTML Elements

An HTML element usually consists of a **start** tag and an **end** tag, with the content inserted in between:

<tagname>Content goes here...</tagname>

The HTML **element** is everything from the start tag to the end tag:

My first paragraph.

Start tag	Element content	End tag
<h1>	My First Heading	</h1>
	My first paragraph.	

HTML elements with no content are called empty elements. Empty elements do not have an end tag, such as the
 element (which indicates a line break).

Nested HTML Elements

HTML elements can be nested (elements can contain elements).

All HTML documents consist of nested HTML elements.

This example contains four HTML elements:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Example Explained

The `<html>` element defines the **whole document**.

It has a **start** tag <html> and an **end** tag </html>.

Inside the <html> element is the `<body>` element.

```
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

The `<body>` element defines the **document body**.

It has a **start** tag <body> and an **end** tag </body>.

Inside the <body> element is two other HTML elements: `<h1>` and `<p>`.

```
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
```

The `<h1>` element defines a **heading**.

It has a **start** tag <h1> and an **end** tag </h1>.

The element **content** is: My First Heading.

```
<h1>My First Heading</h1>
```

The `<p>` element defines a **paragraph**.

It has a **start** tag and an **end** tag .

The element **content** is: My first paragraph.

```
<p>My first paragraph.</p>
```

Do Not Forget the End Tag

Some HTML elements will display correctly, even if you forget the end tag:

Example

```
<html>
<body>

<p>This is a paragraph
<p>This is a paragraph

</body>
</html>
```

The example above works in all browsers, because the closing tag is considered optional.

Never rely on this. It might produce unexpected results and/or errors if you forget the end tag.

Empty HTML Elements

HTML elements with no content are called empty elements.

`<br>` is an empty element without a closing tag (the `<br>` tag defines a line break):

Example

```
<p>This is a <br> paragraph with a line break.</p>
```

Empty elements can be "closed" in the opening tag like this:
.

HTML5 does not require empty elements to be closed. But if you want stricter validation, or if you need to make your document readable by XML parsers, you must close all HTML elements properly.

HTML Is Not Case Sensitive

HTML tags are not case sensitive: <P> means the same as <p>.

The HTML5 standard does not require lowercase tags, but W3C **recommends** lowercase in HTML, and **demands** lowercase for stricter document types like XHTML.

HTML Attributes

Attributes provide additional information about HTML elements.

HTML Attributes

- All HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes usually come in name/value pairs like: **name="value"**

The href Attribute

HTML links are defined with the `<a>` tag. The link address is specified in the `href` attribute:

Example

```
<a href="https://www.xyzmyschools.com">This is a link</a>
```

The src Attribute

HTML images are defined with the `<img>` tag.

The filename of the image source is specified in the `src` attribute:

Example

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

The width and height Attributes

HTML images also have `width` and `height` attributes, which specifies the width and height of the image:

Example

```
<img src="img_girl.jpg" width="500" height="600">
```

The width and height are specified in pixels by default; so width="500" means 500 pixels wide.

You will learn more about images in our HTML Images chapter.

The alt Attribute

The `alt` attribute specifies an alternative text to be used, if an image cannot be displayed.

The value of the `alt` attribute can be read by screen readers. This way, someone "listening" to the webpage, e.g. a vision impaired person, can "hear" the element.

Example

```
<img src="img_girl.jpg" alt="Girl with a jacket">
```

The `alt` attribute is also useful if the image cannot be displayed (e.g. if it does not exist):

Example

See what happens if we try to display an image that does not exist:

```
<img src="img_typo.jpg" alt="Girl with a jacket">
```

The style Attribute

The `style` attribute is used to specify the styling of an element, like color, font, size etc.

Example

```
<p style="color:red">This is a red paragraph.</p>
```

The lang Attribute

The language of the document can be declared in the `<html>` tag.

The language is declared with the `lang` attribute.

Declaring a language is important for accessibility applications (screen readers) and search engines:

```
<!DOCTYPE html>
<html lang="en-US">
<body>

...

</body>
</html>
```

The first two letters specify the language (en). If there is a dialect, add two more letters (US).

The title Attribute

Here, a `title` attribute is added to the `<p>` element. The value of the title attribute will be displayed as a tooltip when you mouse over the paragraph:

Example

```
<p title="I'm a tooltip">
This is a paragraph.
</p>

Here is HTML code for Reference:

<!DOCTYPE html>

<html>

<body>

<h2 title="I'm a header">The title Attribute</h2>

<p title="I'm a tooltip">

Mouse over this paragraph, to display the title attribute as a tooltip.

</p></body></html>
```

We Suggest: Use Lowercase Attributes

The HTML5 standard does not require lowercase attribute names.

The title attribute can be written with uppercase or lowercase
like **title** or **TITLE**.

W3C **recommends** lowercase in HTML, and **demands** lowercase for stricter document types like XHTML.

We Suggest: Quote Attribute Values

The HTML5 standard does not require quotes around attribute values.

The `href` attribute, demonstrated above, *can* be written without quotes:

Single or Double Quotes?

Double quotes around attribute values are the most common in HTML, but single quotes can also be used.

In some situations, when the attribute value itself contains double quotes, it is necessary to use single quotes:

```
<p title='John "ShotGun" Nelson'>
```

Or vice versa:

```
<p title="John 'ShotGun' Nelson">
```

Chapter Summary

- All HTML elements can have **attributes**
- The `title` attribute provides additional "tool-tip" information
- The `href` attribute provides address information for links
- The `width` and `height` attributes provide size information for images
- The `alt` attribute provides text for screen readers

HTML Headings

Headings

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

HTML Code for Reference:

```
<!DOCTYPE html>

<html>

<body>


<h1>Heading 1</h1>

<h2>Heading 2</h2>

<h3>Heading 3</h3>

<h4>Heading 4</h4>

<h5>Heading 5</h5>

<h6>Heading 6</h6>
```

```
</body></html>
```

HTML Headings

Headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading.

Note: Browsers automatically add some white space (a margin) before and after a heading.

Headings Are Important

Search engines use the headings to index the structure and content of your web pages.

Users often skim a page by its headings. It is important to use headings to show the document structure.

`<h1>` headings should be used for main headings, followed by `<h2>` headings, then the less important `<h3>`, and so on.

Note: Use HTML headings for headings only. Don't use headings to make text **BIG** or **bold**.

Bigger Headings

Each HTML heading has a default size. However, you can specify the size for any heading with the `style` attribute, using the CSS `font-size` property:

Example

```
<h1 style="font-size:60px;">Heading 1</h1>
```

HTML Horizontal Rules

The `<hr>` tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.

The `<hr>` element is used to separate content (or define a change) in an HTML page:

```
<h1>This is heading 1</h1>
<p>This is some text.</p>
<hr>
<h2>This is heading 2</h2>
<p>This is some other text.</p>
<hr>
```

The HTML <head> Element

The HTML `<head>` element is a container for metadata. HTML metadata is data about the HTML document. Metadata is not displayed.

The `<head>` element is placed between the `<html>` tag and the `<body>` tag:

```
<!DOCTYPE html>
<html>

<head>
  <title>My First HTML</title>
  <meta charset="UTF-8">
</head>

<body>
```

.

. **Note:** Metadata typically define the document title, character set, styles, scripts, and other meta information

How to View HTML Source?

Have you ever seen a Web page and wondered "Hey! How did they do that?"

View HTML Source Code:

Right-click in an HTML page and select "View Page Source" (in Chrome) or "View Source" (in Edge), or similar in other browsers. This will open a window containing the HTML source code of the page.

Inspect an HTML Element:

Right-click on an element (or a blank area), and choose "Inspect" or "Inspect Element" to see what elements are made up of (you will see both the HTML and the CSS). You can also edit the HTML or CSS on-the-fly in the Elements or Styles panel that opens.

HTML Paragraphs

HTML Paragraphs

The HTML `<p>` element defines a **paragraph**:

Example

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

```
HTML sample given below for Reference:

<!DOCTYPE html>

<html>

<body>

<p>This is a paragraph.</p>

<p>This is a paragraph.</p>

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

```
</body>

</html>
```

HTML Display

You cannot be sure how HTML will be displayed.

Large or small screens, and resized windows will create different results.

With HTML, you cannot change the output by adding extra spaces or extra lines in your HTML code.

The browser will remove any extra spaces and extra lines when the page is displayed:

Example

```
<p>
This paragraph
contains a lot of lines
in the source code,
but the browser
ignores it.
</p>

<p>
This paragraph
contains         a lot of spaces
in the source          code,
but the        browser
ignores it.
</p>
```

Don't Forget the End Tag

Most browsers will display HTML correctly even if you forget the end tag:

```
<p>This is a paragraph.
<p>This is another paragraph.
```

Note: Dropping the end tag can produce unexpected results or errors.

HTML Line Breaks

The HTML `<br>` element defines a **line break**.

Use `<br>` if you want a line break (a new line) without starting a new paragraph:

Example

```
<p>This is<br>a paragraph<br>with line breaks.</p>
```

The HTML <pre> Element

The HTML `<pre>` element defines preformatted text.

The text inside a `<pre>` element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks:

```
<pre>
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.
</pre>
```

HTML Images

Images can improve the design and the appearance of a web page.

Example

```
<img src="pic_trulli.jpg" alt="Italian Trulli">

HTML Sample

<!DOCTYPE html>

<html>

<body>

<h2>HTML Image</h2>

<img src="pic_trulli.jpg" alt="Trulli" width="500" height="333">

</body>

</html>
```

HTML Images Syntax

In HTML, images are defined with the `<img>` tag.

The `<img>` tag is empty, it contains attributes only, and does not have a closing tag.

The `src` attribute specifies the URL (web address) of the image:

```
<img src="url">
```

The alt Attribute

The `alt` attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

The value of the `alt` attribute should describe the image:

```
<img src="img_chania.jpg" alt="Flowers in Chania">
```

If a browser cannot find an image, it will display the value of the `alt` attribute:

```
<img src="wrongname.gif" alt="Flowers in Chania">
```

Image Size - Width and Height

You can use the `style` attribute to specify the width and height of an image.

```
<img src="img_girl.jpg" alt="Girl in a
jacket" style="width:500px;height:600px;">
```

Alternatively, you can use the `width` and `height` attributes:

Example

```
<img src="img_girl.jpg" alt="Girl in a jacket" width="500" height="600">
```

Width and Height, or Style?

The `width`, `height`, and `style` attributes are valid in HTML.

However, we suggest using the `style` attribute. It prevents styles sheets from changing the size of images:

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
  width: 100%;
}
</style>
</head>
<body>
<img src="html5.gif" alt="HTML5 Icon" width="128" height="128">
<img src="html5.gif" alt="HTML5 Icon" style="width:128px;height:128px;">
</body>
</html>
```

Images in Another Folder

If not specified, the browser expects to find the image in the same folder as the web page.

However, it is common to store images in a sub-folder. You must then include the folder name in the `src` attribute:

Example

```
<img src="/images/html5.gif" alt="HTML5
Icon" style="width:128px;height:128px;">
```

Images on Another Server

Some web sites store their images on image servers.

Actually, you can access images from any web address in the world:

Example

```
<img src="https://www.xyzmyschools.com/images/xyzmyschools_green.jpg" alt=
"xyzmySchools.com">
```

Animated Images

HTML allows animated GIFs:

Example

```
<img src="programming.gif" alt="Computer
Man" style="width:48px;height:48px;">
```

Image as a Link

To use an image as a link, put the `<img>` tag inside the `<a>` tag:

Example

```
<a href="default.asp">
  <img src="smiley.gif" alt="HTML
tutorial" style="width:42px;height:42px;border:0;">
</a>
```

Note: `border:0;` is added to prevent IE9 (and earlier) from displaying a border around the image (when the image is a link).

Image Floating

Use the CSS `float` property to let the image float to the right or to the left of a text:

Example

```
<p><img src="smiley.gif" alt="Smiley
face" style="float:right;width:42px;height:42px;">
The image will float to the right of the text.</p>

<p><img src="smiley.gif" alt="Smiley
face" style="float:left;width:42px;height:42px;">
The image will float to the left of the text.</p>
```

HTML Screen Readers

A screen reader is a software program that reads the HTML code, converts the text, and allows the user to "listen" to the content. Screen readers are useful for people who are visually impaired or learning disabled.

Chapter Summary

- Use the HTML `<img>` element to define an image

- Use the HTML `src` attribute to define the URL of the image
- Use the HTML `alt` attribute to define an alternate text for an image, if it cannot be displayed
- Use the HTML `width` and `height` attributes to define the size of the image
- Use the CSS `width` and `height` properties to define the size of the image (alternatively)
- Use the CSS `float` property to let the image float

Loading images takes time. Large images can slow down your page. Use images carefully.

HTML Image Maps

With image maps, you can add clickable areas on an image.

The `<map>` tag defines an image-map. An image-map is an image with clickable areas.

Click on the computer, the phone, or the cup of coffee in the image below:

Example

```
<img src="workplace.jpg" alt="Workplace" usemap="#workmap">

<map name="workmap">
  <area shape="rect" coords="34,44,270,350" alt="Computer" href="computer.
htm">
  <area shape="rect" coords="290,172,333,250" alt="Phone" href="phone.htm"
>
  <area shape="circle" coords="337,300,44" alt="Coffee" href="coffee.htm">
</map>
```

How Does it Work?

The idea behind an image map is that you should be able to perform different actions depending on where in the image you click.

To create an image map you need an image, and a map containing some rules that describe the clickable areas.

The Image

The image is inserted using the `<img>` tag. The only difference from other images is that you must add a `usemap` attribute:

```
<img src="workplace.jpg" alt="Workplace" usemap="#workmap">
```

The `usemap` value starts with a hash tag `#` followed by the name of the image map, and is used to create a relationship between the image and the image map.

The Map

Then add a `<map>` element.

The `<map>` element is used to create an image map, and is linked to the image by using the `name` attribute:

```
<map name="workmap">
```

The `name` attribute must have the same value as the `usemap` attribute.

Note: You may insert the `<map>` element anywhere you like, it does not have to be inserted right after the image.

HTML Background Images

To add a background image in HTML, use the CSS property `background-image`.

Background Image on a HTML element

To add a background image on an HTML element, you can use the `style` attribute:

Example

Add a background image on a HTML element:

```
<div style="background-image: url('img_girl.jpg');">
```

You can also specify the background image in the `<style>` *element*:

Example

Specify the background image in the style element:

```
<style>
div {
  background-image: url('img_girl.jpg');
}
</style>
```

Background Image on a Page

If you want the entire page to have a background image, then you must specify the background image on the `<body>` element:

Example

Add a background image on a HTML page:

```
<style>
body {
  background-image: url('img_girl.jpg');
}
</style>
```

Background Repeat

If the background image is smaller than the element, the image will repeat itself, horizontally and vertically, until it has reach the end of the element.

To explain, see what happens when we use a small image as a background inside a large div element:

The `background-image` property will try to fill the div element with images until it has reach the end.

Example

```
<style>
body {
  background-image: url('example_img_girl.jpg');
}
</style>
```

To avoid the background image from repeating itself, use the `background-repeat` property.

HTML Picture Element

The picture element allows us to display different pictures for different devices or screen sizes.

The HTML <picture> Element

HTML5 introduced the `<picture>` element to add more flexibility when specifying image resources.

The `<picture>` element contains a number of `<source>` elements, each referring to different image sources. This way the browser can choose the image that best fits the current view and/or device.

Each `<source>` element have attributes describing when their image is the most suitable.

Example

Show different images on different screen sizes:

```
<picture>
  <source media="(min-width: 650px)" srcset="img_food.jpg">
  <source media="(min-width: 465px)" srcset="img_car.jpg">
```

```
  <img src="img_girl.jpg">
</picture>
```

Note: Always specify an `<img>` element as the last child element of the `<picture>` element. The `<img>` element is used by browsers that do not support the `<picture>` element, or if none of the `<source>` tags matched.

When to use the Picture Element

There are two main purposes for the `<picture>` element:

1. Bandwidth

If you have a small screen or device, it is not necessary to load a large image file. The browser will use the first `<source>` element with matching attribute values, and ignore any of the following elements.

2. Format Support

Some browsers or devices may not support all image formats. By using the `<picture>` element, you can add images of all formats, and the browser will use the first format it recognizes and ignore any of the following.

Example

The browser will use the first image format it recognizes:

```
<picture>
  <source srcset="img_avatar.png">
  <source srcset="img_girl.jpg">
  <img src="img_beatles.gif" alt="Beatles" style="width:auto;">
</picture>
```

Note: The browser will use the first `<source>` element with matching attribute values, and ignore any following `<source>` elements.

HTML Links

Links are found in nearly all web pages. Links allow users to click their way from page to page

HTML Links - Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

Note: A link does not have to be text. It can be an image or any other HTML element.

HTML Links - Syntax

Hyperlinks are defined with the HTML `<a>` tag:

```
<a href="url">link text</a>
```

Example

```
<a href="https://www.xyzmyschools.com/html/">Visit our HTML tutorial</a>
```

The `href` attribute specifies the destination address (https://www.xyzmyschools.com/html/) of the link.

The **link text** is the visible part (Visit our HTML tutorial).

Clicking on the link text will send you to the specified address.

Note: Without a forward slash at the end of subfolder addresses, you might generate two requests to the server. Many servers will automatically add a forward slash to the end of the address, and then create a new request.

Local Links

The example above used an absolute URL (a full web address).

A local link (link to the same web site) is specified with a relative URL (without https://www....).

Example

```
<a href="html_images.asp">HTML Images</a>
```

HTML Links - The target Attribute

The `target` attribute specifies where to open the linked document.

The `target` attribute can have one of the following values:

- `_blank` - Opens the linked document in a new window or tab
- `_self` - Opens the linked document in the same window/tab as it was clicked (this is default)
- `_parent` - Opens the linked document in the parent frame
- `_top` - Opens the linked document in the full body of the window
- *framename* - Opens the linked document in a named frame

This example will open the linked document in a new browser window/tab:

Example

```
<a href="https://www.xyzmySchools.com/" target="_blank">Visit
xyzmySchools!</a>
```

Tip: If your webpage is locked in a frame, you can use `target="_top"` to break out of the frame:

HTML Links - Image as a Link

It is common to use images as links:

Example

```
<a href="default.asp">
  <img src="smiley.gif" alt="HTML
tutorial" style="width:42px;height:42px;border:0;">
</a>
```

Note: `border:0;` is added to prevent IE9 (and earlier) from displaying a border around the image (when the image is a link).

Button as a Link

To use an HTML button as a link, you have to add some JavaScript code.

JavaScript allows you to specify what happens at certain events, such as a click of a button:

Example

```
<button onclick="document.location = 'default.asp'">HTML Tutorial</button>
```

Link Titles

The `title` attribute specifies extra information about an element. The information is most often shown as a tooltip text when the mouse moves over the element.

Example

```
<a href="https://www.xyzmySchools.com/html/" title="Go to xyzmySchools
HTML section">Visit our HTML Tutorial</a>
```

External Paths

External pages can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a web page:

Example

```
<a href="https://www.xyzmySchools.com/html/default.asp">HTML tutorial</a>
```

This example links to a page located in the html folder on the current web site:

HTML Link Colors

HTML Link Colors

By default, a link will appear like this (in all browsers):

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

You can change the default colors, by using CSS:

Example

```
<style>
a:link {
  color: green;
  background-color: transparent;
  text-decoration: none;
}

a:visited {
  color: pink;
  background-color: transparent;
  text-decoration: none;
}
```

```
a:hover {
  color: red;
  background-color: transparent;
  text-decoration: underline;
}

a:active {
  color: yellow;
  background-color: transparent;
  text-decoration: underline;
}
</style>
```

A link can also be styled as a button, by using CSS:

Example

```
<style>
a:link, a:visited {
  background-color: #f44336;
  color: white;
  padding: 15px 25px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
}

a:hover, a:active {
  background-color: red;
}
</style>
```

HTML Link Bookmarks

HTML bookmarks are used to allow readers to jump to specific parts of a Web page.

Bookmarks can be useful if a webpage is very long.

To create a bookmark - first create the bookmark, then add a link to it.

When the link is clicked, the page will scroll down or up to the location with the bookmark.

Example

First, create a bookmark with the `id` attribute:

```
<h2 id="C4">Chapter 4</h2>
```

Then, add a link to the bookmark ("Jump to Chapter 4"), from within the same page:

Example

```
<a href="#C4">Jump to Chapter 4</a>
```

You can also add a link to a bookmark on another page:

HTML Semantic Elements

Semantic elements = elements with a meaning.

What are Semantic Elements?

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of **non-semantic** elements: `<div>` and `<span>` - Tells nothing about its content.

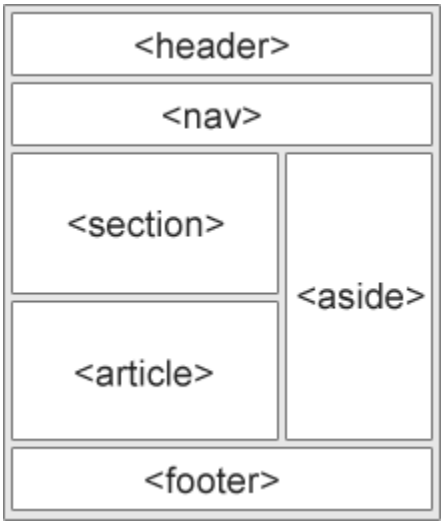
Examples of **semantic** elements: `<form>`, `<table>`, and `<article>` - Clearly defines its content.

Semantic Elements in HTML

Many web sites contain HTML code like: to indicate navigation, header, and footer.

In HTML there are some semantic elements that can be used to define different parts of a web page:

- <article>
- <aside>
- <details>
- <figcaption>
- <figure>
- <footer>
- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>
- <time>



HTML <section> Element

The `<section>` element defines a section in a document.

According to W3C's HTML documentation: "A section is a thematic grouping of content, typically with a heading."

A home page could normally be split into sections for introduction, content, and contact information.

Example

```
<section>
  <h1>WWF</h1>
  <p>The World Wide Fund for Nature (WWF) is....</p>
</section>
```

HTML <article> Element

The `<article>` element specifies independent, self-contained content.

An article should make sense on its own, and it should be possible to read it independently from the rest of the web site.

Examples of where an `<article>` element can be used:

- Forum post
- Blog post
- Newspaper article

Example

```
<article>
  <h1>What Does WWF Do?</h1>
  <p>WWF's mission is to stop the degradation of our planet's natural
environment,
  and build a future in which humans live in harmony with nature.</p>
</article>
```

Nesting <article> in <section> or Vice Versa?

The`<article>` element specifies independent, self-contained content.

The `<section>` element defines section in a document.

Can we use the definitions to decide how to nest those elements? No, we cannot!

So, on the Internet, you will find HTML pages with `<section>` elements containing `<article>` elements, and `<article>` elements containing `<section>` elements.

You will also find pages with `<section>` elements containing `<section>` elements, and `<article>` elements containing `<article>` elements.

Example for a newspaper: The sport `<article>` in the sport **section**, may have a technical **section** in each `<article>`.

HTML <header> Element

The `<header>` element specifies a header for a document or section.

The `<header>` element should be used as a container for introductory content.

You can have several `<header>` elements in one document.

The following example defines a header for an article:

Example

```
<article>
  <header>
    <h1>What Does WWF Do?</h1>
    <p>WWF's mission:</p>
  </header>
  <p>WWF's mission is to stop the degradation of our planet's natural
environment,
  and build a future in which humans live in harmony with nature.</p>
</article>
```

HTML <footer> Element

The `<footer>` element specifies a footer for a document or section.

A `<footer>` element should contain information about its containing element.

A footer typically contains the author of the document, copyright information, links to terms of use, contact information, etc.

You may have several `<footer>` elements in one document.

Example

```
<footer>
  <p>Posted by: Hege Refsnes</p>
  <p>Contact information: <a href="mailto:someone@example.com">
  someone@example.com</a>.</p>
</footer>
```

HTML <nav> Element

The `<nav>` element defines a set of navigation links.

Notice that NOT all links of a document should be inside a `<nav>` element.
The `<nav>` element is intended only for major block of navigation links.

Example

```
<nav>
  <a href="/html/">HTML</a> |
  <a href="/css/">CSS</a> |
  <a href="/js/">JavaScript</a> |
  <a href="/jquery/">jQuery</a>
</nav>
```

HTML <aside> Element

The `<aside>` element defines some content aside from the content it is placed in (like a sidebar).

The `<aside>` content should be related to the surrounding content.

Example

```
<p>My family and I visited The Epcot center this summer.</p>

<aside>
  <h4>Epcot Center</h4>
  <p>The Epcot Center is a theme park in Disney World, Florida.</p>
</aside>
```

HTML <figure> and <figcaption> Elements

An image and a caption can be grouped together in a `<figure>` element.

The purpose of a caption is to add a visual explanation to an image.

Example

```
<figure>
  <img src="pic_trulli.jpg" alt="Trulli">
  <figcaption>Fig1. - Trulli, Puglia, Italy.</figcaption>
</figure>
```

The `<img>` element defines the image, the `<figcaption>` element defines the caption.

Why Semantic Elements?

According to the W3C: "A semantic Web allows data to be shared and reused across applications, enterprises, and communities."

Semantic Elements in HTML

Below is an alphabetical list of some of the semantic elements in HTML.

Tag	Description
<article>	Defines an article
<aside>	Defines content aside from the page content
<details>	Defines additional details that the user can view or hide
<figcaption>	Defines a caption for a <figure> element
<figure>	Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for a document or section
<header>	Specifies a header for a document or section
<main>	Specifies the main content of a document
<mark>	Defines marked/highlighted text

<nav>	Defines navigation links
<section>	Defines a section in a document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time

Adding CSS into HTML Styles

Styling HTML with CSS

CSS stands for **C**ascading **S**tyle **S**heets.

CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**.

CSS **saves a lot of work**. It can control the layout of multiple web pages all at once.

CSS can be added to HTML elements in 3 ways:

- **Inline** - by using the style attribute in HTML elements
- **Internal** - by using a `<style>` element in the `<head>` section
- **External** - by using an external CSS file

The most common way to add CSS, is to keep the styles in separate CSS files. However, here we will use inline and internal styling, because this is easier to demonstrate, and easier for you to try it yourself.

Tip: **You can learn much more about CSS in our CSS Tutorial.**

Inline CSS

An inline CSS is used to apply a unique style to a single HTML element.

An inline CSS uses the style attribute of an HTML element.

This example sets the text color of the `<h1>` element to blue:

Example

```
<h1 style="color:blue;">This is a Blue Heading</h1>
```

Internal CSS

An internal CSS is used to define a style for a single HTML page.

An internal CSS is defined in the `<head>` section of an HTML page, within a `<style>` element:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: powderblue;}
h1   {color: blue;}
p    {color: red;}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

External CSS

An external style sheet is used to define the style for many HTML pages.

With an external style sheet, you can change the look of an entire web site, by changing one file!

To use an external style sheet, add a link to it in the `<head>` section of the HTML page:

Example

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

An external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.

Here is how the "styles.css" looks:

```
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```

CSS Fonts

The CSS `color` property defines the text color to be used.

The CSS `font-family` property defines the font to be used.

The CSS `font-size` property defines the text size to be used.

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
  color: blue;
  font-family: verdana;
  font-size: 300%;
}
p  {
  color: red;
  font-family: courier;
  font-size: 160%;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

CSS Border

The CSS `border` property defines a border around an HTML element:

Example

```
p {
  border: 1px solid powderblue;
}
```

CSS Padding

The CSS `padding` property defines a padding (space) between the text and the border:

<html>

```
<head>
<style>
p {
  border: 1px solid powderblue;
  padding: 30px;
}
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

CSS Margin

The CSS `margin` property defines a margin (space) outside the border:

Example

```
p {
  border: 1px solid powderblue;
  margin: 50px;
}
```

The id Attribute

To define a specific style for one special element, add an `id` attribute to the element:

```
<p id="p01">I am different</p>
```

then define a style for the element with the specific id:

Example

```
#p01 {
  color: blue;
}
```

The class Attribute

To define a style for special types of elements, add a `class` attribute to the element:

```
<p class="error">I am different</p>
```

then define a style for the elements with the specific class:

Example

```
p.error {
  color: red;
}
```

External References

External style sheets can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a style sheet:

Example

```
<html>

<head>

  <link rel="stylesheet" href="https://www.w3schools.com/html/styles.css">

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>
```

Introduction to CSS

CSS is a language that describes the style of an HTML document.

CSS describes how HTML elements should be displayed.

This tutorial will teach you CSS from basic to advanced.

Examples in Each Chapter

This CSS tutorial contains hundreds of CSS examples.

With our online editor, you can edit the CSS, and click on a button to view the result.

CSS Example

```
body {
  background-color: lightblue;
}
```

```css
h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

What is CSS?

- **CSS** stands for **C**ascading **S**tyle **S**heets
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
- External stylesheets are stored in **CSS files**

CSS Demo - One HTML Page - Multiple Styles!

Why Use CSS?

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

CSS Example

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

CSS Solved a Big Problem

HTML was NEVER intended to contain tags for formatting a web page!

HTML was created to **describe the content** of a web page, like:

<h1>This is a heading</h1>

This is a paragraph.

When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.

To solve this problem, the World Wide Web Consortium (W3C) created CSS.

CSS removed the style formatting from the HTML page!

CSS Saves a Lot of Work!

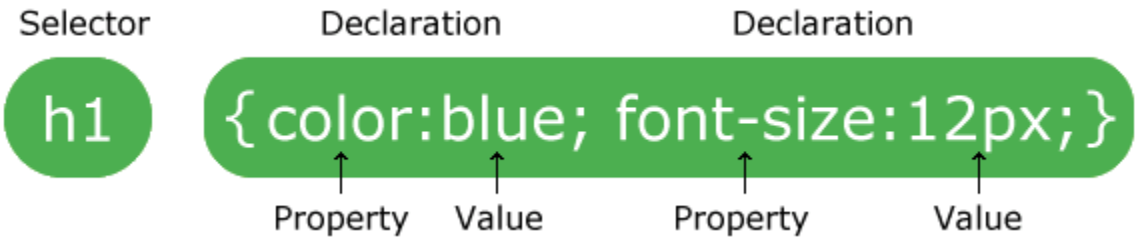
The style definitions are normally saved in external .css files.

With an external stylesheet file, you can change the look of an entire website by changing just one file!

CSS Syntax

CSS Syntax

A CSS rule-set consists of a selector and a declaration block:



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS property name and a value, separated by a colon.

A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.

Example

In this example all <p> elements will be center-aligned, with a red text color:

```
p {
  color: red;
  text-align: center;
}
```

Example Explained

- `p` is a **selector** in CSS (it points to the HTML element you want to style: <p>).
- `color` is a property, and `red` is the property value

- `text-align` is a property, and `center` is the property value

You will learn much more about CSS selectors and CSS properties in the next chapters.

CSS Selectors

```
p {
  text-align: center;
  color: red;
}
```

The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example

The CSS rule below will be applied to the HTML element with id="para1":

```
#para1 {
  text-align: center;
  color: red;
}
```

Note: An id name cannot start with a number!

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

Example

In this example all HTML elements with class="center" will be red and center-aligned:

```
.center {
  text-align: center;
  color: red;
}
```

You can also specify that only specific HTML elements should be affected by a class.

Example

In this example only <p> elements with class="center" will be center-aligned:

```
p.center {
  text-align: center;
  color: red;
}
```

HTML elements can also refer to more than one class.

Example

In this example the <p> element will be styled according to class="center" and to class="large":

```
<p class="center large">This paragraph refers to two classes.</p>
```

The CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

Example

The CSS rule below will affect every HTML element on the page:

```
* {
  text-align: center;
```

```
  color: blue;
}
```

The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

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

h2 {
  text-align: center;
  color: red;
}

p {
  text-align: center;
  color: red;
}
```

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

Example

In this example we have grouped the selectors from the code above:

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

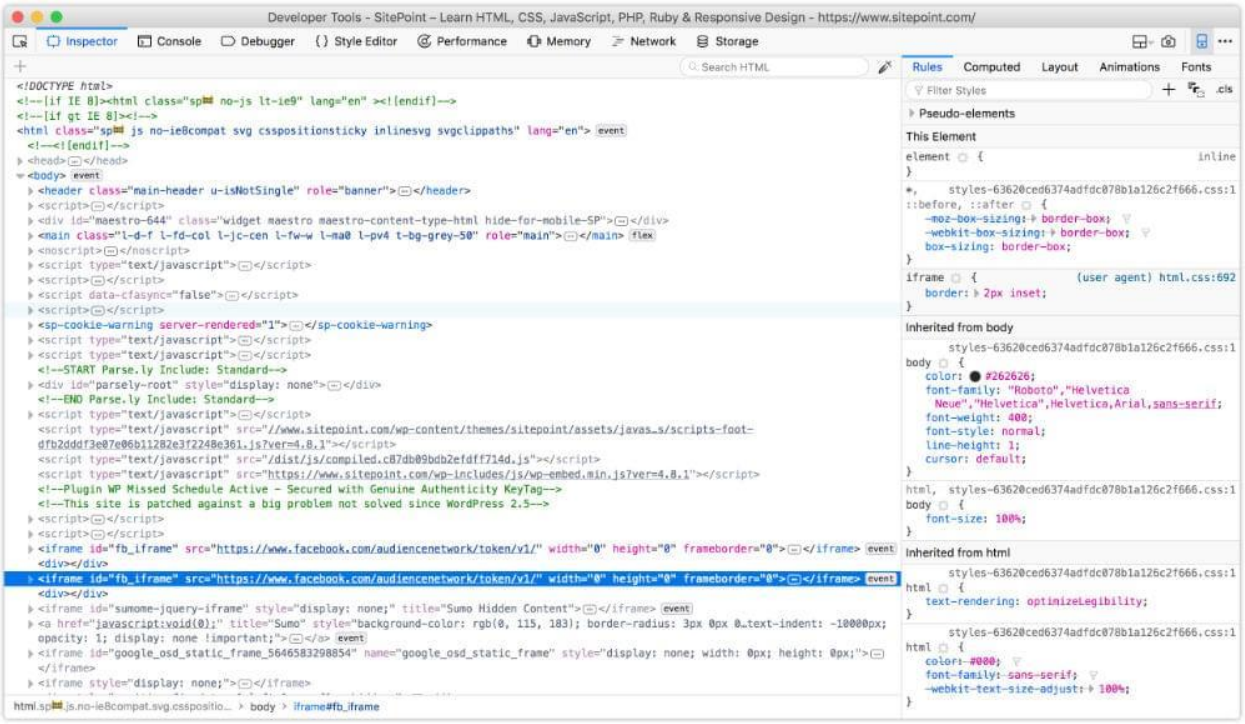
All CSS Simple Selectors

Selector	Example	Example description
.class	.intro	Selects all elements with class="intro"
#id	#firstname	Selects the element with id="firstname"
*	*	Selects all elements
element	p	Selects all <p> elements
element,element,..	div, p	Selects all elements and all <p> elements

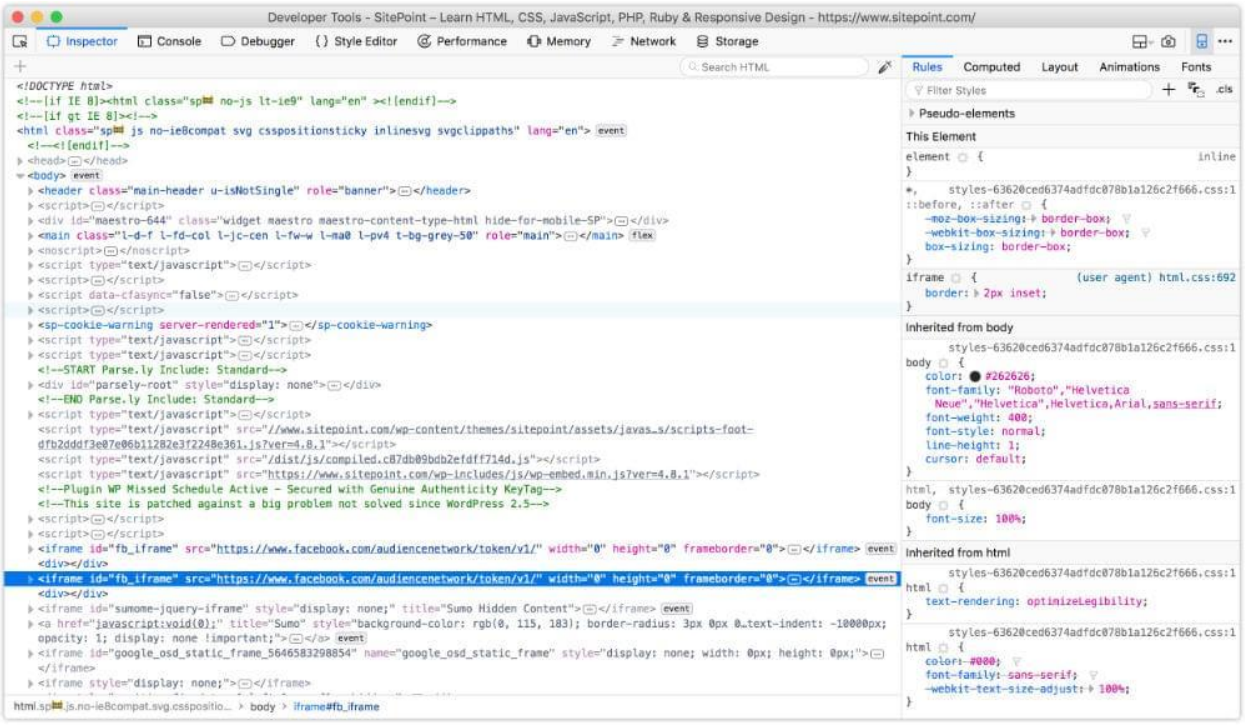
CSS Debugging and Optimization: Browser-based Developer Tools

Browser-based Developer Tools

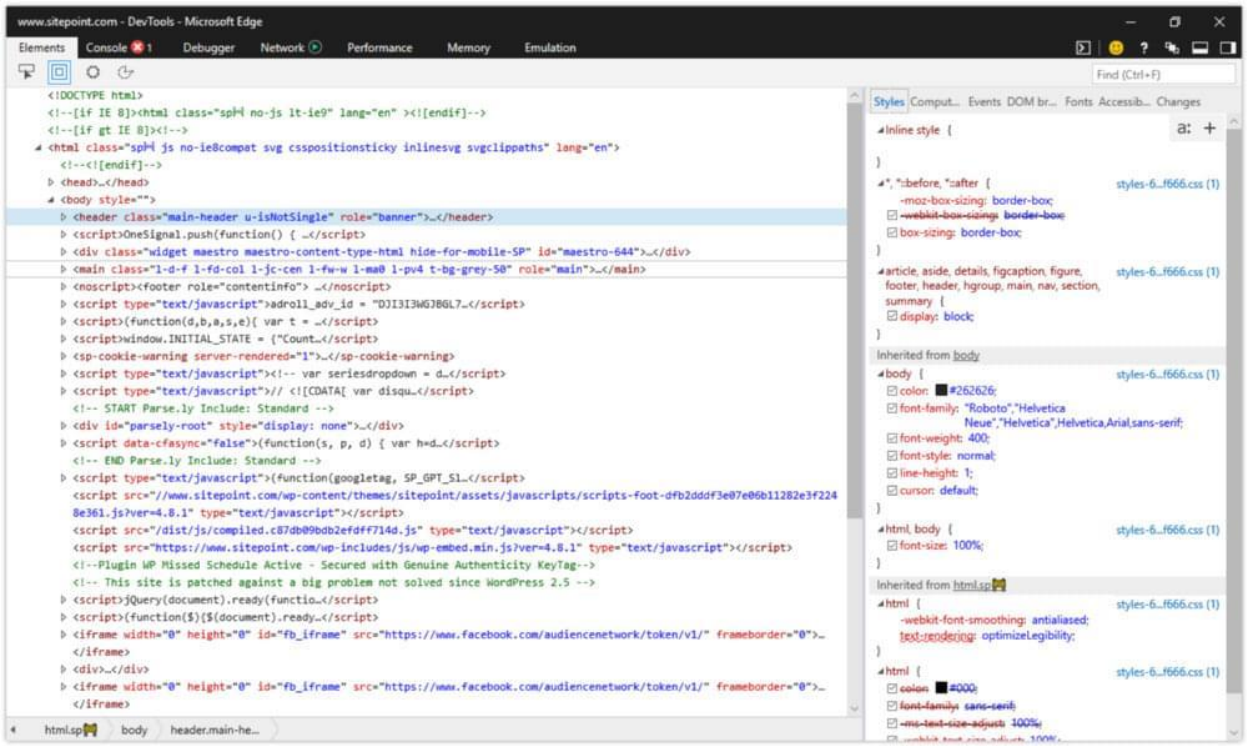
Most desktop browsers include an element inspector feature that you can use to troubleshoot your CSS. Start using this feature by right-clicking and selecting Inspect Element from the menu. Mac users can also inspect an element by clicking the element while pressing the Ctrl key. The image below indicates what you can expect to see in Firefox Developer Edition.



In Firefox, Chrome and Safari you can also press Ctrl + Shift + I (Windows/Linux) or Cmd + Option + I (macOS) to open the developer tools panel. The image below shows the Chrome developer tools.



While in Microsoft Edge, open developer tools by pressing the F12 key, as seen in below.



You can also open each browser's developer tools using the application's menu:

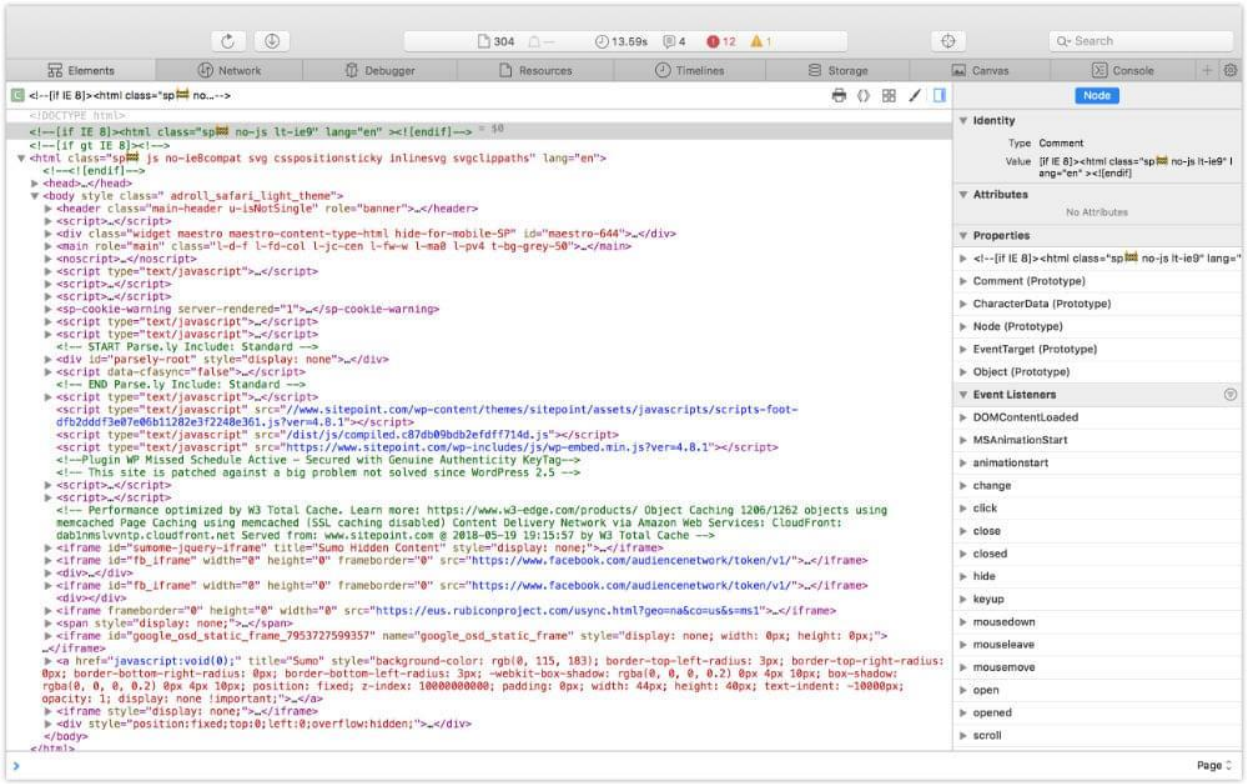
Microsoft Edge: Tools > Developer Tools

Firefox: Tools > Web Developer

Chrome: View > Developer

Safari: Develop > Show Web Inspector

In Safari, you may have to enable the Develop menu first by going to Safari > Preferences… > Advanced and checking the box next to Show Develop menu in menu bar. The view for Safari developer tools is illustrated below.



After opening the developer tools interface, you may then need to select the correct panel:

Microsoft Edge: DOM Explorer

Firefox: Inspector

Chrome: Elements

Safari: Elements

You'll know you're in the right place when you see HTML on one side of the panel, and CSS rules on the other.

Note: The markup you'll see in the HTML panel is a representation of the DOM. It's generated when the browser finishes parsing the document and may differ from

your original markup. Using View Source reveals the original markup, but keep in mind that for JavaScript applications there may not be any markup to view.

Using the Styles Panel

Sometimes an element isn't styled as expected. Maybe a typographical change failed to take, or there's less padding around a paragraph than you wanted. You can determine which rules are affecting an element by using the Styles panel of the Web Inspector.

Browsers are fairly consistent in how they organize the Styles panel. Inline styles, if any, are typically listed first. These are styles set using the style attribute of HTML, whether by the CSS author or programmatically via scripting.

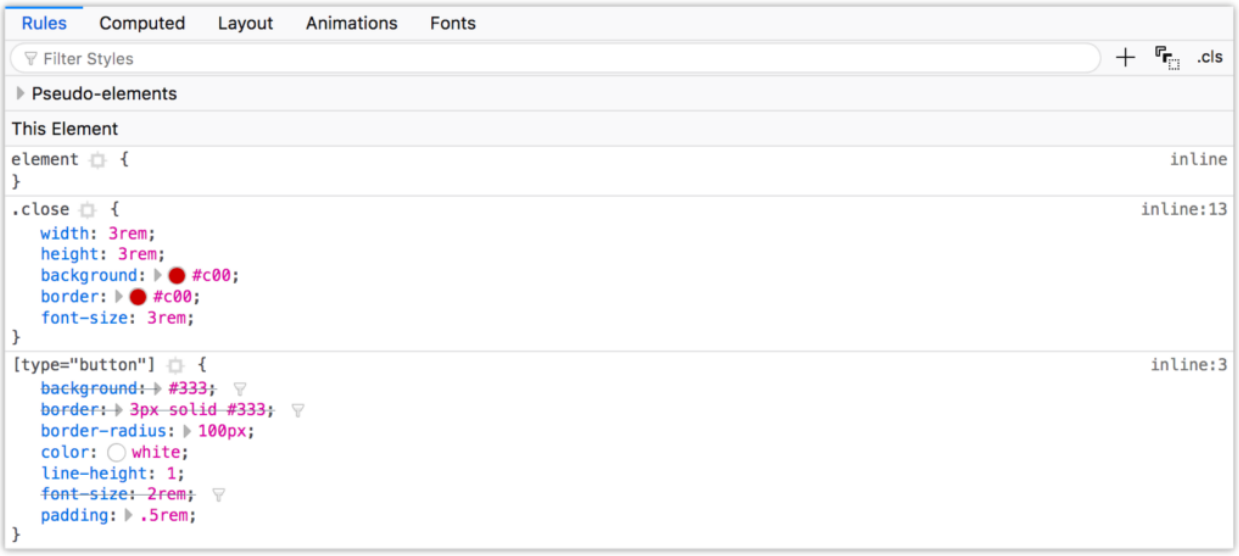
Inline styles are followed by a list of style rules applied via author stylesheets—those written by you or your colleagues. Styles in this list are grouped by media query and/or filename.

Authored style rules precede user agent styles. User agent styles are the browser's default styles. They too have an impact on your site's look and feel. (In Firefox, you may have to select the **Show Browser Styles** option in order to view user agent styles. You can find this setting in the Toolbox Options panel.)

Properties and values are grouped by selector. A checkbox sits next to each property, letting you toggle specific rules on and off. Clicking on a property or value allows you to change it, so you can avoid having to edit, save and reload.

Identifying Cascade and Inheritance Problems

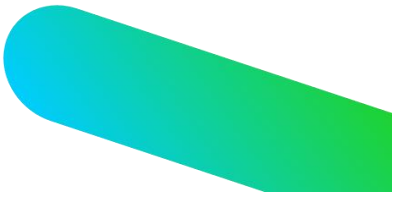
As you inspect styles, you may notice that some properties appear crossed out. These properties have been overridden either by a cascading rule, a conflicting rule, or a more specific selector, as depicted below.

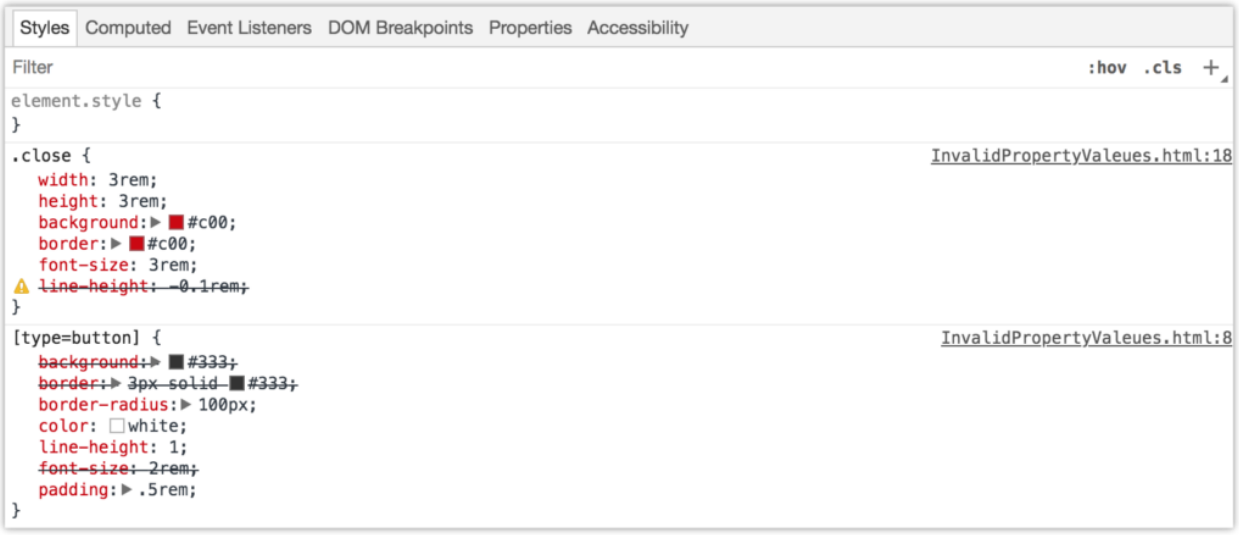


In the image above, the background, border, and font-size declarations of the [type=button] block are displayed with a line through them. These declarations were overridden by those in the .close block, which succeeds the [type=button] in our CSS.

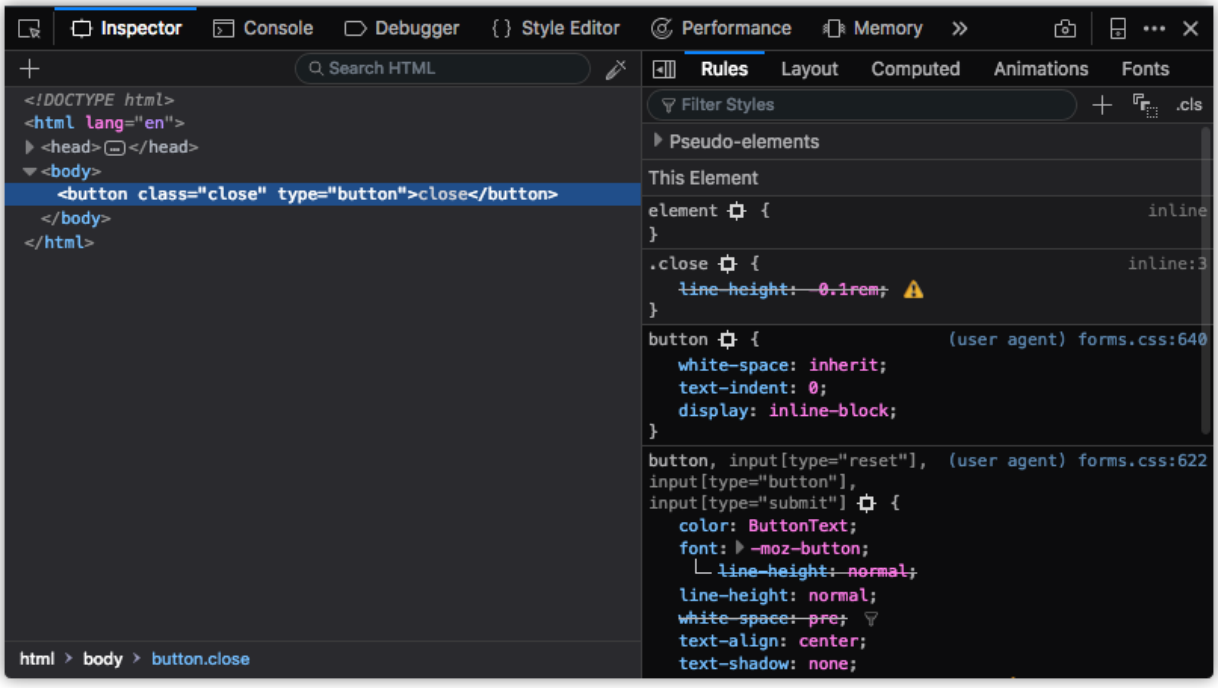
Spotting Invalid Properties and Values

You can also use the element inspector to spot invalid or unsupported properties and property values. In Chromium-based browsers, invalid CSS rules both have a line through them and an adjacent warning icon, which can be seen below.

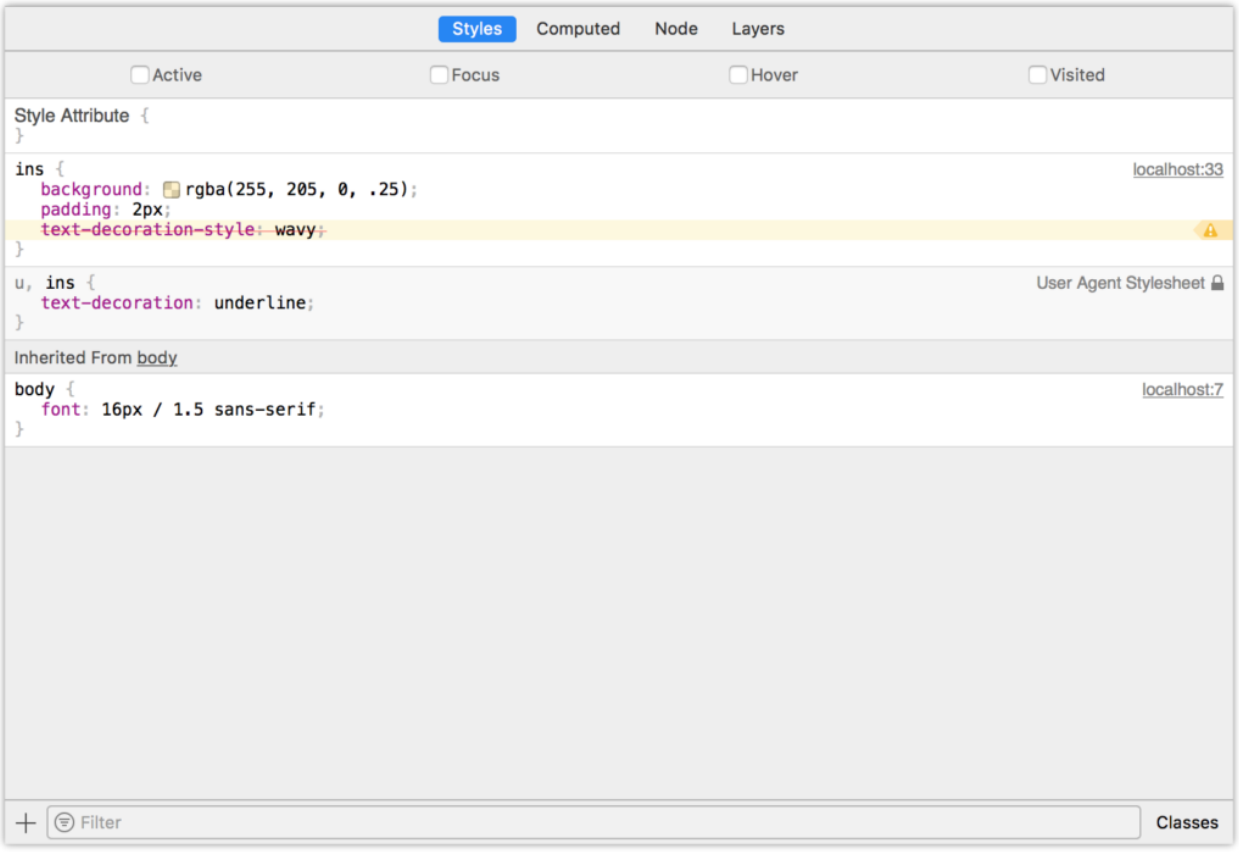




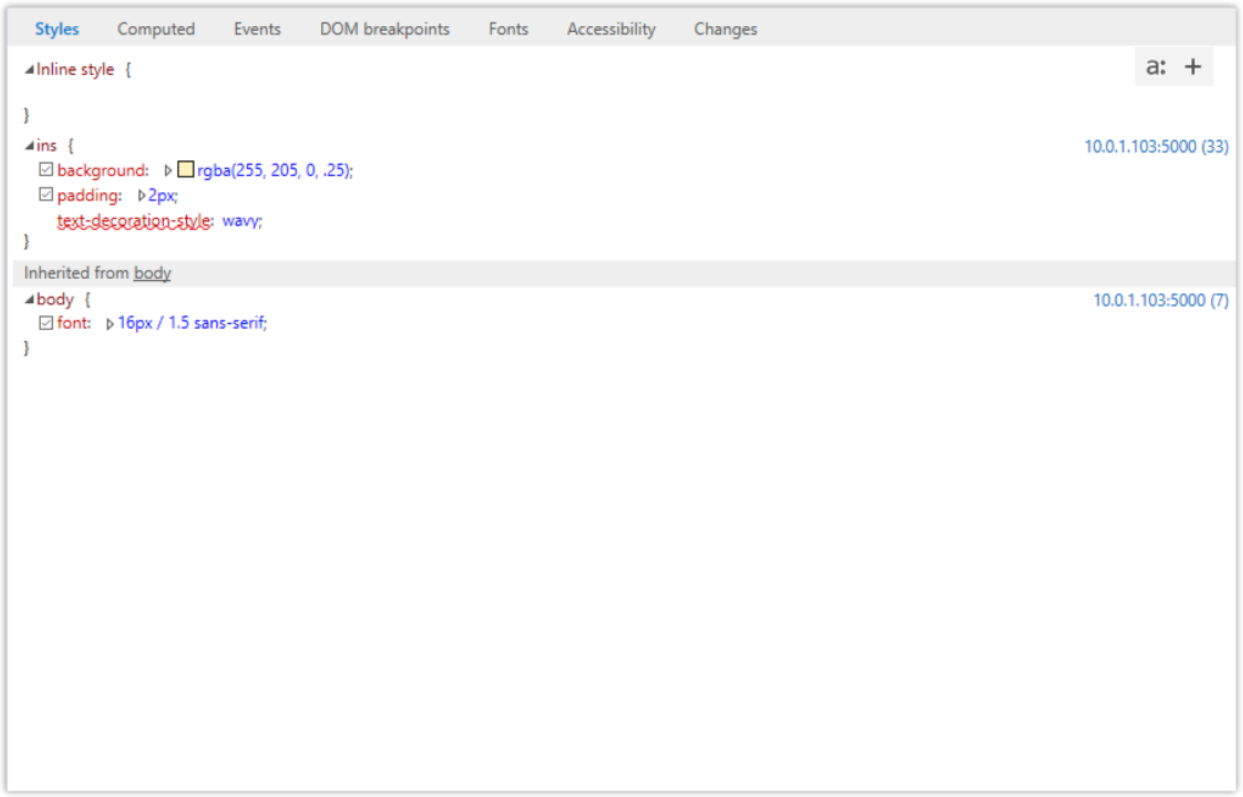
Firefox also strikes through invalid or unsupported properties and values. Firefox Developer Edition also uses a warning icon, as shown below. Standard Firefox displays errors similarly, but doesn't include the warning icon.



In the screenshot below, Safari strikes through unsupported rules with a red line, and highlights them with a yellow background and warning icon.



Microsoft Edge instead uses a wavy underline to indicate unsupported properties or values.



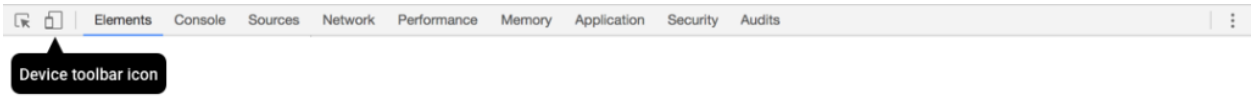
When it comes to basic debugging and inheritance conflicts, whichever browser you choose doesn't matter. Familiarize yourself with all of them, however, for those rare occasions when you need to diagnose a browser-specific issue.

Debugging Responsive Layouts

On-device testing is always best. During development, however, it's helpful to simulate mobile devices with your desktop browser. All major desktop browsers include a mode for responsive debugging.

Chrome

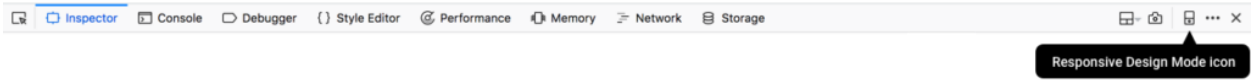
Chrome offers a device toolbar feature as part of its developer toolkit. To use it, click the device icon (pictured below) in the upper-left corner, next to the Select an element icon.



Device mode lets you mimic several kinds of Android and iOS devices, including older devices such as the iPhone 5 and Galaxy S5. Device mode also includes a network throttling feature for approximating different network speeds, and the ability to simulate being offline.

Firefox

In Firefox, the equivalent mode is known as **Responsive Design Mode**. Its icon resembles early iPods. You'll find it on the right side of the screen, in the developer tools panel, as shown below.



In responsive mode, you can toggle between portrait and landscape orientations, simulate touch events, and capture screenshots. Like Chrome, Firefox also allows developers to simulate slow connections via throttling.

Precedence in CSS

On your average CSS-writin' day, odds are you won't even think about precedence in CSS. It doesn't come up a whole heck of a lot. But it does matter! It comes up any time multiple CSS selectors match an element with the exact same specificity.

Within a single stylesheet

Say we have some HTML like this:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="module module-foo module-bar">
    Module
```

```
  </div>
</body>

</html>
```

The order of the attributes in the HTML *don't* matter. It's the order in the stylesheet. Let's focus on the `background`:

```
/*
  All of these selectors match
  and they all have the same specificity
*/

.module {
  background: #ccc;
  padding: 20px;
  max-width: 400px;
  margin: 20px auto;
}

.module-foo {
  background: orange;
}

/* LAST declared, so these property/values win */
.module-bar {
  background: lightblue; /* I win! */

  /* I still have all the _other_ styles as well */

}
```

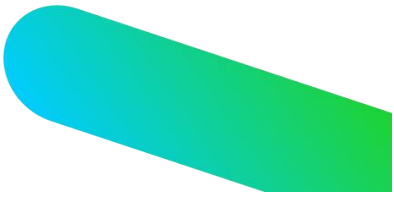
An intentionally convoluted example

Order is not limited to a single stylesheet. The order of the stylesheet in the document matters even more.

Check out this document with three distinct style... uh... let's call them chunks. A chunk being either a <link rel="stylesheet">, a <style> block, or an @imported stylesheet.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>

  <!-- #1 -->
  <link rel="stylesheet" href="1.css">

  <!-- #2 -->
  <style>
    .module-baz {
      background-color: pink;
    }
  </style>

</head>
<body>
  <div class="module module-foo module-bar module-baz">
    Module
  </div>

  <!-- #3 -->
  <style>
    @import "2.css";
    /*
      Contains
      .module-bar { background: #f06d06; }
    */
  </style>

</body>

</html>
```

I labeled the chunks #1, #2, and #3. All of them contain CSS selectors with the same exact specificity. #3 is the last declared, so it wins the precedence battle.



Async loaded CSS still respects document order

Perhaps you're loading CSS with an awesome CSS loader like loadCSS. What happens if we were to load a fourth CSS file with it with the exact same setup as the "convoluted" example above?

loadCSS injects the stylesheet at the bottom of the <head> by default, so it would become #3 and the <style> block at the bottom of the body would become #4 and thus win.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <link rel="stylesheet" href="1.css">

  <script src="loadCSS.js"></script>
  <script>
    loadCSS("2.css");
  </script>

  <!-- 2.css will be injected right here -->
</head>
<body>
  <div class="module module-foo module-bar module-late">
    Module
  </div>
</body>

</html>
```

It's actually invalid (although it works) to have a <link> or <style> block as a child of the <body>, so it would be really rare for a stylesheet loaded by loadCSS to not be the winner by default.

Also, you can specify an ID to target so you can control CSS order:

```
<script id="loadcss">
  // load a CSS file just before the script element containing this
code
  loadCSS("path/to/mystylesheet.css",
document.getElementById("loadcss"));

</script>
```

CSS Media Queries

CSS2 Introduced Media Types

The `@media` rule, introduced in CSS2, made it possible to define different style rules for different media types.

Examples: You could have one set of style rules for computer screens, one for printers, one for handheld devices, one for television-type devices, and so on.

Unfortunately these media types never got a lot of support by devices, other than the print media type.

CSS3 Introduced Media Queries

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

- width and height of the viewport
- width and height of the device
- orientation (is the tablet/phone in landscape or portrait mode?)
- resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

Media Query Syntax

A media query consists of a media type and can contain one or more expressions, which resolve to either true or false.

```
@media not|only mediatype and (expressions) {
  CSS-Code;
}
```

The result of the query is true if the specified media type matches the type of device the document is being displayed on and all expressions in the media query are true. When a media query is true, the corresponding style sheet or style rules are applied, following the normal cascading rules.

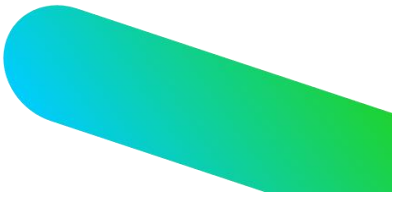
Unless you use the not or only operators, the media type is optional and the `all` type will be implied.

You can also have different stylesheets for different media:

```
<link rel="stylesheet" media="mediatype and|not|only
(expressions)" href="print.css">
```

CSS3 Media Types

Value	Description
all	Used for all media type devices
print	Used for printers
screen	Used for computer screens, tablets, smart-phones etc.
speech	Used for screenreaders that "reads" the page out loud



Media Queries Simple Examples

One way to use media queries is to have an alternate CSS section right inside your style sheet.

The following example changes the background-color to lightgreen if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the background-color will be pink):

Example

```
@media screen and (min-width: 480px) {
  body {
    background-color: lightgreen;
  }
}
```

The following example shows a menu that will float to the left of the page if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the menu will be on top of the content):

Example

```
@media screen and (min-width: 480px) {
  #leftsidebar {width: 200px; float: left;}
  #main {margin-left: 216px;}
}
```

CSS @media Rule

Definition and Usage

The `@media` rule is used in media queries to apply different styles for different media types/devices.

Media queries can be used to check many things, such as:

- width and height of the viewport

- width and height of the device
- orientation (is the tablet/phone in landscape or portrait mode?)
- resolution

Using media queries are a popular technique for delivering a tailored style sheet (responsive web design) to desktops, laptops, tablets, and mobile phones.

You can also use media queries to specify that certain styles are only for printed documents or for screen readers (mediatype: print, screen, or speech).

In addition to media types, there are also media features. Media features provide more specific details to media queries, by allowing to test for a specific feature of the user agent or display device. For example, you can apply styles to only those screens that are greater, or smaller, than a certain width.

CSS Syntax

meaning of the **not**, **only** and **and** keywords:

not: The not keyword reverts the meaning of an entire media query.

only: The only keyword prevents older browsers that do not support media queries with media features from applying the specified styles. **It has no effect on modern browsers.**

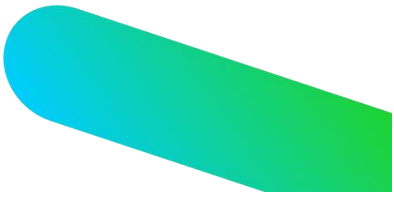
and: The and keyword combines a media feature with a media type or other media features.

They are all optional. However, if you use **not** or **only**, you must also specify a media type.

You can also have different *stylesheets* for different media, like this:

Media Types

Value	Description
all	Default. Used for all media type devices



print	Used for printers
screen	Used for computer screens, tablets, smart-phones etc.
speech	Used for screenreaders that "reads" the page out loud

CSS Website Layout

Website Layout

A website is often divided into headers, menus, content and a footer:

There are tons of different layout designs to choose from. However, the structure above, is one of the most common, and we will take a closer look at it in this tutorial.

Header

A header is usually located at the top of the website (or right below a top navigation menu). It often contains a logo or the website name:

Example

```
.header {
  background-color: #F1F1F1;
  text-align: center;
  padding: 20px;
}
```

Navigation Bar

A navigation bar contains a list of links to help visitors navigating through your website:

Example

```
/* The navbar container */
.topnav {
  overflow: hidden;
  background-color: #333;
}

/* Navbar links */
.topnav a {
  float: left;
  display: block;
  color: #f2f2f2;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

/* Links - change color on hover */
.topnav a:hover {
  background-color: #ddd;
  color: black;
}
```

Content

The layout in this section, often depends on the target users. The most common layout is one (or combining them) of the following:

- **1-column** (often used for mobile browsers)
- **2-column** (often used for tablets and laptops)
- **3-column layout** (only used for desktops)

- We will create a 3-column layout, and change it to a 1-column layout on smaller screens:

- **Example**

```
/* Create three equal columns that floats next to each other */
.column {
  float: left;
  width: 33.33%;
}

/* Clear floats after the columns */
.row:after {
  content: "";
  display: table;
  clear: both;
}

/* Responsive layout - makes the three columns stack on top of each
other instead of next to each other on smaller screens (600px wide or
less) */
@media screen and (max-width: 600px) {
  .column {
    width: 100%;
  }
}
```

Tip: To create a 2-column layout, change the width to 50%. To create a 4-column layout, use 25%, etc.

Tip: A more modern way of creating column layouts, is to use CSS Flexbox. However, it is not supported in Internet Explorer 10 and earlier versions. If you require IE6-10 support, use floats (as shown above).

Unequal Columns

The main content is the biggest and the most important part of your site.

It is common with **unequal** column widths, so that most of the space is reserved for the main content. The side content (if any) is often used as an alternative navigation or to specify information relevant to the main content. Change the widths as you like, only remember that it should add up to 100% in total:

Example

```
.column {
  float: left;
}

/* Left and right column */
.column.side {
  width: 25%;
}

/* Middle column */
.column.middle {
  width: 50%;
}

/* Responsive layout - makes the three columns stack on top of each other
instead of next to each other */
@media screen and (max-width: 600px) {
  .column.side, .column.middle {
    width: 100%;
  }
}
```

Footer

The footer is placed at the bottom of your page. It often contains information like copyright and contact info:

Example

```
.footer {
  background-color: #F1F1F1;
  text-align: center;
  padding: 10px;
}
```

JavaScript

JavaScript is the programming language of HTML and the Web.

JavaScript is easy to learn.

```
<!DOCTYPE html>
<html>
<body>
<h2>My First JavaScript</h2>
<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button>
<p id="demo"></p>
</body>
</html>
```

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`.

This example uses the method to "find" an HTML element (with id="demo") and changes the element content (`innerHTML`) to "Hello JavaScript":

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change HTML content.</p>
```

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>

</body>

</html>

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the `src` (source) attribute of an `<img>` tag:

Example

<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the light</button>

</body>

</html>

JavaScript Variables

JavaScript variables are containers for storing data values.

In this example, `x`, `y`, and `z`, are variables:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Variables</h2>
<p>In this example, x, y, and z are variables.</p>
<p id="demo"></p>
<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>
</body>
</html>
```

From the example above, you can expect:

- x stores the value 5
- y stores the value 6
- z stores the value 11

JavaScript Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Assignment Operators

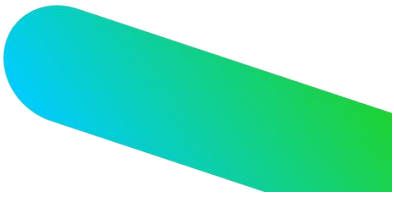
Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y

-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator



JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Example

Calculate the product of two numbers, and return the result:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a calculation and returns the result:</p>

<p id="demo"></p>

<script>

var x = myFunction(4, 3);

document.getElementById("demo").innerHTML = x;


function myFunction(a, b) {

  return a * b;

}

</script>

</body>

</html>
```

JavaScript Examples

JavaScript Expressions

```
<html>

<body>
```

```
<h2>JavaScript Expressions</h2>

<p>Expressions compute to values.</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML = 5 * 10;

</script>

</body>

</html>
```

What Can JavaScript Do?

```
<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick='document.getElementById("demo").innerHTML
= "Hello JavaScript!"'>Click Me!</button>

</body>

</html>
```

JavaScript can change the style of an HTML element

```
<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button"
onclick="document.getElementById('demo').style.fontSize='35px'">Click
Me!</button>
```

```
</body>
</html>
```

JavaScript Objects

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p>There are two different ways to access an object property.</p>
<p>You can use person.property or person["property"].</p>
<p id="demo"></p>
<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id     :  5566
};
// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " " + person.lastName;
</script>
</body>
</html>
```

JavaScript Events

```
<html>

<body>

<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>

<p id="demo"></p>

</body>

</html>
```

JavaScript HTML DOM Examples

Examples of using JavaScript to access and manipulate DOM objects.

```
<html>

<body>

<p>This document was last modified <span id="demo"></span>.</p>

<script>

document.getElementById("demo").innerHTML = document.lastModified;

</script>

</body>

</html>
```

JavaScript Conditionals

```
<!DOCTYPE html>

<html>

<body>

<p>Click the button to get a time-based greeting:</p>
```

```
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var greeting;
  var time = new Date().getHours();
  if (time < 10) {
    greeting = "Good morning";
  } else if (time < 20) {
    greeting = "Good day";
  } else {
    greeting = "Good evening";
  }
  document.getElementById("demo").innerHTML = greeting;
}
</script>
</body>
</html>
```

JavaScript Loops

```
<html>
<body>
<h2>JavaScript For/In Loop</h2>
```

```
<p>The for/in statement loops through the properties of an object.</p>
<p id="demo"></p>
<script>
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};
var x;
for (x in person) {
  txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>
</body>
</html>
```

JavaScript Error Handling

```
<html>
<script>
var txt = "";
function message() {
  try {
    adddlert("Welcome guest!");
  }
  catch(err) {
    txt = "There was an error on this page.\n\n";
```

```
    txt += "Click OK to continue viewing this page,\n";
    txt += "or Cancel to return to the home page.\n\n";
    if(!confirm(txt)) {
      document.location.href = "https://www.w3schools.com/";
    }
  }
}
</script>
<body>
<h2>JavaScript Error Handling</h2>
<input type="button" value="View message" onclick="message()" />
</body>
</html>
```

JavaScript Regular Expressions

```
<html>
<body>
<h2>JavaScript Regular Expressions</h2>
<p>Search a string for "mySchools", and display the position of the match:</p>
<p id="demo"></p>
<script>
var str = "Visit MySchools!";
var n = str.search(/w3Schools/i);
document.getElementById("demo").innerHTML = n;
```

```
</script>

</body>

</html>
```

JavaScript String Concatenation

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Operators</h2>

<p>The assignment operator += can concatenate strings.</p>

<p id="demo"></p>

<script>

txt1 = "What a very ";

txt1 += "nice day";

document.getElementById("demo").innerHTML = txt1;

</script>

</body>

</html>
```

JavaScript Strings

```
<!DOCTYPE html>

<html>

<body>
```

```
<p>Click the button to perfom a global (/g) search for the letters "ain" (/ain) in a string, and display the matches.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

function myFunction() {

  var str = "The rain in SPAIN stays mainly in the plain";

  var res = str.match(/ain/g);

  document.getElementById("demo").innerHTML = res;

}

</script>

</body>

</html>
```

JavaScript Number Methods

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Number Methods</h2>

<p>The toPrecision() method returns a string, with a number written with a specified length:</p>

<p id="demo"></p>

<script>

var x = 9.656;

document.getElementById("demo").innerHTML =
```

```
  x.toPrecision() + "<br>" +
  x.toPrecision(2) + "<br>" +
  x.toPrecision(4) + "<br>" +
  x.toPrecision(6);
</script>
</body>
</html>
```

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`:

Example

```
<html>
<body>

<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

In the example above, `getElementById` is a **method**, while `innerHTML` is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the `id` of the element.

In the example above the `getElementById` method used `id="demo"` to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the `innerHTML` property.

The `innerHTML` property is useful for getting or replacing the content of HTML elements.

The `innerHTML` property can be used to get or change any HTML element, including `<html>` and `<body>`.

Playing with HTML 5

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.

HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

Browser Support

The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5 functionality.

The mobile web browsers that come pre-installed on iPhones, iPads, and Android phones all have excellent support for HTML5.

New Features

HTML5 introduces a number of new elements and attributes that can help you in building modern websites. Here is a set of some of the most prominent features introduced in HTML5.

- New Semantic Elements: These are like <header>, <footer>, and <section>.

- Forms 2.0: Improvements to HTML web forms where new attributes have been introduced for <input> tag.

- Persistent Local Storage: To achieve without resorting to third-party plugins.

- WebSocket : A next-generation bidirectional communication technology for web applications.

- Server-Sent Events: HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).

- Canvas: This supports a two-dimensional drawing surface that you can program with JavaScript.

- Audio & Video: You can embed audio or video on your webpages without resorting to third-party plugins.

HTML5

- Geolocation: Now visitors can choose to share their physical location with your web application.

- Microdata: This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.

- Drag and drop: Drag and drop the items from one location to another location on the same webpage.

Backward Compatibility

HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. Its new features have been built on existing features and allow you to provide fallback content for older browsers.

It is suggested to detect support for individual HTML5 features using a few lines of JavaScript.

If you are not familiar with any previous version of HTML, I would recommend that you go through our HTML Tutorial before exploring the features of HTML5.

HTML5 Geolocation

The HTML Geolocation API is used to locate a user's position.

Locate the User's Position

The HTML Geolocation API is used to get the geographical position of a user.

Since this can compromise privacy, the position is not available unless the user approves it.

Note: Geolocation is most accurate for devices with GPS, like smartphone.

Note: As of Chrome 50, the Geolocation API will only work on secure contexts such as HTTPS. If your site is hosted on an non-secure origin (such as HTTP) the requests to get the users location will no longer function.

Using HTML Geolocation

The `getCurrentPosition()` method is used to return the user's position.

The example below returns the latitude and longitude of the user's position:

Example

```
<script>
var x = document.getElementById("demo");
```

```
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}

function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
```

Example explained:

- Check if Geolocation is supported
- If supported, run the getCurrentPosition() method. If not, display a message to the user
- If the getCurrentPosition() method is successful, it returns a coordinates object to the function specified in the parameter (showPosition)
- The showPosition() function outputs the Latitude and Longitude

The example above is a very basic Geolocation script, with no error handling.

Handling Errors and Rejections

The second parameter of the `getCurrentPosition()` method is used to handle errors. It specifies a function to run if it fails to get the user's location:

Example

```
function showError(error) {
  switch(error.code) {
    case error.PERMISSION_DENIED:
      x.innerHTML = "User denied the request for Geolocation."
      break;
    case error.POSITION_UNAVAILABLE:
      x.innerHTML = "Location information is unavailable."
      break;
    case error.TIMEOUT:
      x.innerHTML = "The request to get user location timed out."
      break;
    case error.UNKNOWN_ERROR:
```

```
      x.innerHTML = "An unknown error occurred."
      break;
  }
}
```

Displaying the Result in a Map

To display the result in a map, you need access to a map service, like Google Maps.

In the example below, the returned latitude and longitude is used to show the location in a Google Map (using a static image):

Example

```
function showPosition(position) {
  var latlon = position.coords.latitude + "," + position.coords.longitude;

  var img_url = "https://maps.googleapis.com/maps/api/staticmap?center=
  "+latlon+"&zoom=14&size=400x300&sensor=false&key=YOUR_KEY";

  document.getElementById("mapholder").innerHTML = "<img
src='"+img_url+"'>";
}
```

Location-specific Information

This page has demonstrated how to show a user's position on a map.

Geolocation is also very useful for location-specific information, like:

- Up-to-date local information
- Showing Points-of-interest near the user
- Turn-by-turn navigation (GPS)

The getCurrentPosition() Method - Return Data

The `getCurrentPosition()` method returns an object on success. The latitude, longitude and accuracy properties are always returned. The other properties are returned if available:

Property	Returns
coords.latitude	The latitude as a decimal number (always returned)
coords.longitude	The longitude as a decimal number (always returned)
coords.accuracy	The accuracy of position (always returned)
coords.altitude	The altitude in meters above the mean sea level (returned if available)
coords.altitudeAccuracy	The altitude accuracy of position (returned if available)
coords.heading	The heading as degrees clockwise from North (returned if available)
coords.speed	The speed in meters per second (returned if available)
timestamp	The date/time of the response (returned if available)

Geolocation Object - Other interesting Methods

The Geolocation object also has other interesting methods:

- `watchPosition()` - Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).
- `clearWatch()` - Stops the `watchPosition()` method.

The example below shows the `watchPosition()` method. You need an accurate GPS device to test this (like smartphone):

Example

```
<script>
var x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.watchPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
```

HTML5 Drag and Drop

Drag and Drop

Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

In HTML5, drag and drop is part of the standard: Any element can be draggable.

HTML Drag and Drop Example

The example below is a simple drag and drop example:

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev) {
  ev.preventDefault();
```

```
}

function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>

<img id="drag1" src="img_logo.gif" draggable="true" ondragstart="drag(even
t)" width="336" height="69">

</body>
</html>
```

It might seem complicated, but lets go through all the different parts of a drag and drop event.

Make an Element Draggable

First of all: To make an element draggable, set the `draggable` attribute to true:

```
<img draggable="true">
```

What to Drag - ondragstart and setData()

Then, specify what should happen when the element is dragged.

In the example above, the `ondragstart` attribute calls a function, drag(event), that specifies what data to be dragged.

The `dataTransfer.setData()` method sets the data type and the value of the dragged data:

```
function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}
```

In this case, the data type is "text" and the value is the id of the draggable element ("drag1").

Where to Drop - ondragover

The `ondragover` event specifies where the dragged data can be dropped.

By default, data/elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element.

This is done by calling the `event.preventDefault()` method for the ondragover event:

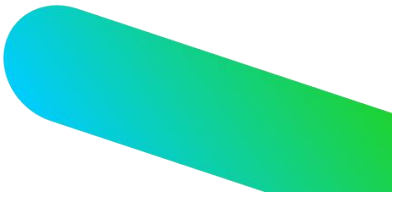
```
event.preventDefault()
```

Do the Drop - ondrop

When the dragged data is dropped, a drop event occurs.

In the example above, the ondrop attribute calls a function, drop(event):

```
function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
```



Code explained:

- Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop)
- Get the dragged data with the dataTransfer.getData() method. This method will return any data that was set to the same type in the setData() method
- The dragged data is the id of the dragged element ("drag1")
- Append the dragged element into the drop element

Array Iteration Methods

Array iteration methods operate on every array item.

Array.forEach()

The `forEach()` method calls a function (a callback function) once for each array element.

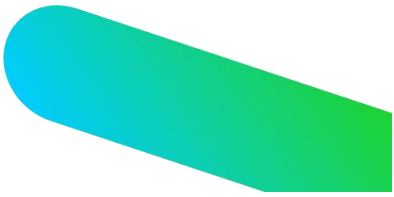
Example

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

The example above uses only the value parameter. The example can be rewritten to:



Example

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value) {
  txt = txt + value + "<br>";
}
```

Array.map()

The `map()` method creates a new array by performing a function on each array element.

The `map()` method does not execute the function for array elements without values.

The `map()` method does not change the original array.

This example multiplies each array value by 2:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses only the value parameter, the index and array parameters can be omitted:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value) {
  return value * 2;
}
```

Array.filter()

The `filter()` method creates a new array with array elements that passes a test.

This example creates a new array from elements with a value larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

In the example above, the callback function does not use the index and array parameters, so they can be omitted:

Example

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value) {
  return value > 18;
}
```

`Array.filter()` is supported in all browsers except Internet Explorer 8 or earlier.

Array.reduce()

The `reduce()` method runs a function on each array element to produce (reduce it to) a single value.

The `reduce()` method works from left-to-right in the array. See also `reduceRight()`.

The `reduce()` method does not reduce the original array.

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value, index, array) {
  return total + value;
}
```

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

Array.reduceRight()

The `reduceRight()` method runs a function on each array element to produce (reduce it to) a single value.

The `reduceRight()` works from right-to-left in the array. See also `reduce()`.

The `reduceRight()` method does not reduce the original array.

This example finds the sum of all numbers in an array:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduceRight(myFunction);
```

```
function myFunction(total, value, index, array) {
  return total + value;
}
```

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

Array.every()

The `every()` method check if all array values pass a test.

This example check if all array values are larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses the first parameter only (value), the other parameters can be omitted:

Example

```
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value) {
```

```
  return value > 18;
}
```

Array.some()

The `some()` method check if some array values pass a test.

This example check if some array values are larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var someOver18 = numbers.some(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

`Array.some()` is supported in all browsers except Internet Explorer 8 or earlier.

Array.indexOf()

The `indexOf()` method searches an array for an element value and returns its position.

Note: The first item has position 0, the second item has position 1, and so on.

Example

Search an array for the item "Apple":

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");
```

Syntax

array.indexOf(*item*, *start*)

item	Required. The item to search for.
start	Optional. Where to start the search. Negative values will start at the given position cou end, and search to the end.

`Array.indexOf()` returns -1 if the item is not found.

If the item is present more than once, it returns the position of the first occurrence.

Array.lastIndexOf()

`Array.lastIndexOf()` is the same as `Array.indexOf()`, but returns the position of the last occurrence of the specified element.

Example

Search an array for the item "Apple":

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.lastIndexOf("Apple");
```

Syntax

array.lastIndexOf(*item*, *start*)

item Required. The item to search for

start Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the beginning

Array.find()

The `find()` method returns the value of the first array element that passes a test function.

This example finds (returns the value of) the first element that is larger than 18:

Example

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.find(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

`Array.find()` is not supported in older browsers. The first browser versions with full support is listed below.

Array.findIndex()

The `findIndex()` method returns the index of the first array element that passes a test function.

This example finds the index of the first element that is larger than 18:

Example

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.findIndex(myFunction);

function myFunction(value, index, array) {
```

```
  return value > 18;
}
```

HTML Multimedia

Multimedia on the web is sound, music, videos, movies, and animations.

Multimedia comes in many different formats. It can be almost anything you can hear or see.

Examples: Images, music, sound, videos, records, films, animations, and more.

Web pages often contain multimedia elements of different types and formats.

In this chapter you will learn about the different multimedia formats.

Browser Support

The first web browsers had support for text only, limited to a single font in a single color.

Later came browsers with support for colors and fonts, and images!

Audio, video, and animation have been handled differently by the major browsers. Different formats have been supported, and some formats require extra helper programs (plug-ins) to work.

Hopefully this will become history. HTML5 multimedia promises an easier future for multimedia.

Multimedia Formats

Multimedia elements (like audio or video) are stored in media files.

The most common way to discover the type of a file, is to look at the file extension.

Multimedia files have formats and different extensions like: .swf, .wav, .mp3, .mp4, .mpg, .wmv, and .avi.

Common Video Formats

MP4 is the new and upcoming format for internet video.

MP4 is recommended by YouTube.

MP4 is supported by Flash Players.

MP4 is supported by HTML5.

Format	File	Description
MPEG	.mpg	
	.mpeg	MPEG. Developed by the Moving Pictures Expert Group. The first popular video format on the Web. Used to be supported by all browsers, but it is not supported in HTML5 (See MP4).
AVI	.avi	AVI (Audio Video Interleave). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
WMV	.wmv	WMV (Windows Media Video). Developed Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
QuickTime	.mov	QuickTime. Developed by Apple. Commonly used in video cameras and TV hardware. Plays well on Apple computers, but not in web browsers. (See MP4)
RealVideo	.rm	
	.ram	RealVideo. Developed by Real Media to allow video streaming with low bandwidths. It is still used for online video and Internet TV, but does not play in web browsers.
Flash	.swf, .flv	Flash. Developed by Macromedia. Often requires an extra component (plug-in) to play in web browsers.

Ogg .ogg Theora Ogg. Developed by the Xiph.Org Foundation. Supported by HTML5.

WebM .webm WebM. Developed by the web giants, Mozilla, Opera, Adobe, and Google. Supported by HTML5.

MPEG-4

or MP4 .mp4 MP4. Developed by the Moving Pictures Expert Group. Based on QuickTime. Commonly used in newer video cameras and TV hardware. Supported by all HTML5 browsers. Recommended by YouTube.

Only MP4, WebM, and Ogg video are supported by the HTML5 standard.

Audio Formats

MP3 is the newest format for compressed recorded music. The term MP3 has become synonymous with digital music.

If your website is about recorded music, MP3 is the choice.

Format	File	Description
MIDI	.mid .midi	MIDI (Musical Instrument Digital Interface). Main format for all electronic music devices like synthesizers and PC sound cards. MIDI files do not contain sound, but digital notes that can be played by electronics. Plays well on all computers and music hardware, but not in web browsers
RealAudio	.rm .ram	RealAudio. Developed by Real Media to allow streaming of audio with low Does not play in web browsers.

WMA	.wma	WMA (Windows Media Audio). Developed by Microsoft. Commonly used in music players. Plays well on Windows computers, but not in web browsers.
AAC	.aac	AAC (Advanced Audio Coding). Developed by Apple as the default format well on Apple computers, but not in web browsers.
WAV	.wav	WAV. Developed by IBM and Microsoft. Plays well on Windows, Macintosh operating systems. Supported by HTML5.
Ogg	.ogg	Ogg. Developed by the Xiph.Org Foundation. Supported by HTML5.
MP3	.mp3	MP3 files are actually the sound part of MPEG files. MP3 is the most popular format for music players. Combines good compression (small files) with high quality. Supported by all browsers.
MP4	.mp4	MP4 is a video format, but can also be used for audio. MP4 video is the upcoming video format on the internet. This leads to automatic support for MP4 audio by all browsers.

Only MP3, WAV, and Ogg audio are supported by the HTML5 standard.

HTML5 Video

HTML Video Example

```
<html>

<body>

<video width="400" controls>

  <source src="mov_bbb.mp4" type="video/mp4">

  <source src="mov_bbb.ogg" type="video/ogg">

  Your browser does not support HTML5 video.

</video>

<p>

Video courtesy of

<a href="https://www.bigbuckbunny.org/" target="_blank">Big Buck Bunny</a>.

</p>

</body>

</html>
```

Playing Videos in HTML

Before HTML5, a video could only be played in a browser with a plug-in (like flash).

The HTML5 `<video>` element specifies a standard way to embed a video in a web page.

The HTML <video> Element

To show a video in HTML, use the `<video>` element:

Example

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

How it Works

The `controls` attribute adds video controls, like play, pause, and volume.

It is a good idea to always include `width` and `height` attributes. If height and width are not set, the page might flicker while the video loads.

The `<source>` element allows you to specify alternative video files which the browser may choose from. The browser will use the first recognized format.

The text between the `<video>` and `</video>` tags will only be displayed in browsers that do not support the `<video>` element.

HTML <video> Autoplay

To start a video automatically use the `autoplay` attribute:

Example

```
<video width="320" height="240" autoplay>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

The autoplay attribute does not work in mobile devices like iPad and iPhone.

HTML Video - Browser Support

In HTML5, there are 3 supported video formats: MP4, WebM, and Ogg.

The browser support for the different formats is:

Browse	MP4	WebM	Ogg
Internet Explorer	YES	NO	NO
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	NO	NO
Opera	YES (from Opera 25)	YES	YES

HTML Video - Media Types

File Format	Media Type
MP4	video/mp4

WebM	video/webm
Ogg	video/ogg

HTML Video - Methods, Properties, and Events

HTML5 defines DOM methods, properties, and events for the `<video>` element.

This allows you to load, play, and pause videos, as well as setting duration and volume.

There are also DOM events that can notify you when a video begins to play, is paused, etc.

HTML5 Video Tags

Tag	Description
<video>	Defines a video or movie
<source>	Defines multiple media resources for media elements, such as <video> and <audio>
<track>	Defines text tracks in media players

HTML5 Audio

Audio on the Web

Before HTML5, audio files could only be played in a browser with a plug-in (like flash).

The HTML5 `<audio>` element specifies a standard way to embed audio in a web page.

The HTML <audio> Element

To play an audio file in HTML, use the `<audio>` element:

Example

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

HTML Audio - How It Works

The `controls` attribute adds audio controls, like play, pause, and volume.

The `<source>` element allows you to specify alternative audio files which the browser may choose from. The browser will use the first recognized format.

The text between the `<audio>` and `</audio>` tags will only be displayed in browsers that do not support the `<audio>` element.

HTML Audio - Media Types

File Format	Media Type
MP3	audio/mpeg
OGG	audio/ogg
WAV	audio/wav

HTML Audio - Methods, Properties, and Events

HTML5 defines DOM methods, properties, and events for the `<audio>` element.

This allows you to load, play, and pause audios, as well as set duration and volume.

There are also DOM events that can notify you when an audio begins to play, is paused, etc.

HTML5 Audio Tags

Tag	Description
<audio>	Defines sound content
<source>	Defines multiple media resources for media elements, such as <video> and <audio>

HTML Plug-ins

The purpose of a plug-in is to extend the functionality of a web browser.

HTML Helpers (Plug-ins)

Helper applications (plug-ins) are computer programs that extend the standard functionality of a web browser.

Examples of well-known plug-ins are Java applets.

Plug-ins can be added to web pages with the `<object>` tag or the `<embed>` tag.

Plug-ins can be used for many purposes: display maps, scan for viruses, verify your bank id, etc.

To display video and audio: Use the `<video>` and `<audio>` tags.

The <object> Element

The `<object>` element is supported by all browsers.

The `<object>` element defines an embedded object within an HTML document.

It is used to embed plug-ins (like Java applets, PDF readers, Flash Players) in web pages.

Example

```
<object width="400" height="50" data="bookmark.swf"></object>
```

The `<object>` element can also be used to include HTML in HTML:

Example

```
<object width="100%" height="500px" data="snippet.html"></object>
```

The <embed> Element

The `<embed>` element is supported in all major browsers.

The `<embed>` element also defines an embedded object within an HTML document.

Web browsers have supported the <embed> element for a long time. However, it has not been a part of the HTML specification before HTML5.

```
<html>

<body>

<embed width="400" height="50" src="bookmark.swf">

</body>

</html>
```

Note that the <embed> element does not have a closing tag. It can not contain alternative text.

HTML YouTube Videos

The easiest way to play videos in HTML, is to use YouTube.

Struggling with Video Formats?

Earlier in this tutorial, you have seen that you might have to convert your videos to different formats to make them play in all browsers.

Converting videos to different formats can be difficult and time-consuming.

An easier solution is to let YouTube play the videos in your web page.

YouTube Video Id

YouTube will display an id (like tgbNymZ7vqY), when you save (or play) a video.

You can use this id, and refer to your video in the HTML code.

Playing a YouTube Video in HTML

To play your video on a web page, do the following:

- Upload the video to YouTube
- Take a note of the video id
- Define an <iframe> element in your web page
- Let the src attribute point to the video URL
- Use the width and height attributes to specify the dimension of the player
- Add any other parameters to the URL (see below)

```
<html>

<body>

<iframe width="420" height="345"
                    src="https://www.youtube.com/embed/tgbNymZ7vqY">

</iframe>

</body>

</html>
```

YouTube Autoplay

You can have your video start playing automatically when a user visits that page by adding a simple parameter to your YouTube URL.

Note: Take careful consideration when deciding to autoplay your videos. Automatically starting a video can annoy your visitor and end up causing more harm than good.

Value 0 (default): The video will not play automatically when the player loads.

Value 1: The video will play automatically when the player loads.

YouTube - Autoplay

```
<html>

<body>

<iframe width="420" height="345"
                    src="https://www.youtube.com/embed/tgbNymZ7vqY?autopl
                    ay=1">

</iframe>

</body>

</html>
```

JavaScript Objects

In JavaScript, objects are king. If you understand objects, you understand JavaScript.

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the `new` keyword)
- Numbers can be objects (if defined with the `new` keyword)
- Strings can be objects (if defined with the `new` keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives

A **primitive value** is a value that has no properties or methods.

A **primitive data type** is data that has a primitive value.

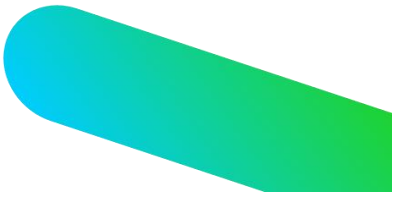
JavaScript defines 5 types of primitive data types:

- `string`
- `number`
- `boolean`
- `null`
- `undefined`

Primitive values are immutable (they are hardcoded and therefore cannot be changed).

if x = 3.14, you can change the value of x. But you cannot change the value of 3.14.

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined



Object Definition

You define (and create) a JavaScript object with an object literal:

Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Spaces and line breaks are not important. An object definition can span multiple lines:

Example

```
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

A method is a function stored as a property.

Example

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

The this Keyword

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the **person object** that "owns" the `fullName` function.

In other words, `this.firstName` means the `firstName` property of **this object**.

Read more about the `this` keyword at JS this Keyword.

Accessing Object Methods

You access an object method with the following syntax:

```
objectName.methodName()
```

Example

```
name = person.fullName();
```

If you access a method **without** the () parentheses, it will return the **function definition**:

Example

```
name = person.fullName;
```

Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "`new`", the variable is created as an object:

```
var x = new String();        // Declares x as a String object
var y = new Number();        // Declares y as a Number object
var z = new Boolean();       // Declares z as a Boolean object
```

Avoid `String`, `Number`, and `Boolean` objects. They complicate your code and slow down execution speed.

JavaScript Object Accessors

JavaScript Accessors (Getters and Setters)

ECMAScript 5 (2009) introduced Getter and Setters.

Getters and setters allow you to define Object Accessors (Computed Properties).

JavaScript Getter (The get Keyword)

This example uses a `lang` property to `get` the value of the `language` property.

```
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  language : "en",
  get lang() {
    return this.language;
  }
```

```
};

// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.lang;
```

JavaScript Setter (The set Keyword)

This example uses a `lang` property to `set` the value of the `language` property.

Example

```
var person = {
  firstName: "John",
  lastName : "Doe",
  language : "",
  set lang(lang) {
    this.language = lang;
  }
};

// Set an object property using a setter:
person.lang = "en";

// Display data from the object:
document.getElementById("demo").innerHTML = person.language;
```

JavaScript Function or Getter?

What is the differences between these two examples?

Example 1

```
var person = {
  firstName: "John",
  lastName : "Doe",
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

```
// Display data from the object using a method:
document.getElementById("demo").innerHTML = person.fullName();
```

Example 2

```
var person = {
  firstName: "John",
  lastName : "Doe",
  get fullName() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object using a getter:
document.getElementById("demo").innerHTML = person.fullName;
```

Example 1 access fullName as a function: person.fullName().

Example 2 access fullName as a property: person.fullName.

The second example provides simpler syntax.

Why Using Getters and Setters?

- It gives simpler syntax
- It allows equal syntax for properties and methods
- It can secure better data quality
- It is useful for doing things behind-the-scenes

Object.defineProperty()

The `Object.defineProperty()` method can also be used to add Getters and Setters:

Example

```
// Define object
var obj = {counter : 0};

// Define setters
Object.defineProperty(obj, "reset", {
  get : function () {this.counter = 0;}
```

```
});
Object.defineProperty(obj, "increment", {
  get : function () {this.counter++;}
});
Object.defineProperty(obj, "decrement", {
  get : function () {this.counter--;}
});
Object.defineProperty(obj, "add", {
  set : function (value) {this.counter += value;}
});
Object.defineProperty(obj, "subtract", {
  set : function (value) {this.counter -= value;}
});

// Play with the counter:
obj.reset;
obj.add = 5;
obj.subtract = 1;
obj.increment;
obj.decrement;
```

JavaScript Object Constructors

Example

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

It is considered good practice to name constructor functions with an upper-case first letter.

Object Types (Blueprints) (Classes)

The examples from the previous chapters are limited. They only create single objects.

Sometimes we need a "**blueprint**" for creating many objects of the same "type".

The way to create an "object type", is to use an **object constructor function**.

In the example above, `function Person()` is an object constructor function.

Objects of the same type are created by calling the constructor function with the `new` keyword:

```
var myFather = new Person("John", "Doe", 50, "blue");
var myMother = new Person("Sally", "Rally", 48, "green");
```

The this Keyword

In JavaScript, the thing called `this` is the object that "owns" the code.

The value of `this`, when used in an object, is the object itself.

In a constructor function `this` does not have a value. It is a substitute for the new object. The value of `this` will become the new object when a new object is created.

Note that `this` is not a variable. It is a keyword. You cannot change the value of `this`.

Adding a Property to an Object

Adding a new property to an existing object is easy:

Example

```
myFather.nationality = "English";
```

The property will be added to myFather. Not to myMother. (Not to any other person objects).

Adding a Method to an Object

Adding a new method to an existing object is easy:

Example

```
<h2>JavaScript Object Constructors</h2>

<p id="demo"></p>

<script>

// Constructor function for Person objects

function Person(first, last, age, eye) {

  this.firstName = first;

  this.lastName = last;

  this.age = age;

  this.eyeColor = eye; }

// Create 2 Person objects

var myFather = new Person("John", "Doe", 50, "blue");

var myMother = new Person("Sally", "Rally", 48, "green");

// Add a name method to first object

myFather.name = function() {

  return this.firstName + " " + this.lastName;

};

// Display full name
```

```
document.getElementById("demo").innerHTML =

"My father is " + myFather.name();

</script>
```

You cannot add a new method to an object constructor the same way you add a new method to an existing object.

Adding methods to an object constructor must be done inside the constructor function:

Built-in JavaScript Constructors

JavaScript has built-in constructors for native objects:

```
var x1 = new Object();    // A new Object object
var x2 = new String();    // A new String object
var x3 = new Number();    // A new Number object
var x4 = new Boolean();   // A new Boolean object
var x5 = new Array();     // A new Array object
var x6 = new RegExp();    // A new RegExp object
var x7 = new Function();  // A new Function object
var x8 = new Date();      // A new Date object
```

HTML Forms

HTML Form Example

```
<html>

<body>

<h2>HTML Forms</h2>

<form action="/action_page.php">

  <label for="fname">First name:</label><br>

  <input type="text" id="fname" name="fname" value="John"><br>
```

```
  <label for="lname">Last name:</label><br>

  <input type="text" id="lname" name="lname" value="Doe"><br><br>

  <input type="submit" value="Submit">

</form>

<p>If you click the "Submit" button, the form-data will be sent to a page called
                    "/action_page.php".</p>

</body>

</html>
```

The <form> Element

The HTML `<form>` element defines a form that is used to collect user input:

```
<form>
                    .
                    form elements
                    .
                    </form>
```

An HTML form contains **form elements**.

Form elements are different types of input elements, like: text fields, checkboxes, radio buttons, submit buttons, and more.

The <input> Element

The `<input>` element is the most important form element.

The `<input>` element is displayed in several ways, depending on the **type** attribute.

Here are some examples:

Type	Description
<input type="text">	Defines a single-line text input field
<input type="radio">	Defines a radio button (for selecting one of many choices)
<input type="submit">	Defines a submit button (for submitting the form)

Text Fields

`<input type="text">` defines a single-line input field for **text input**.

Example

A form with two text input fields:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

The <label> Element

Notice the use of the `<label>` element in the example above.

The `<label>` tag defines a label for many form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out load the label when the user is focused on the input element.

The `<label>` element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the <label> element, it toggles the radio button/checkbox.

The `for` attribute of the <label> tag should be equal to the `id` attribute of the <input> element to bind them together.

Radio Buttons

`<input type="radio">` defines a **radio button**.

Radio buttons let a user select ONE of a limited number of choices.

Example

A form with radio buttons:

```
<form>
  <input type="radio" id="male" name="gender" value="male">
  <label for="male">Male</label><br>
  <input type="radio" id="female" name="gender" value="female">
  <label for="female">Female</label><br>
  <input type="radio" id="other" name="gender" value="other">
  <label for="other">Other</label>
</form>
```

The Submit Button

`<input type="submit">` defines a button for **submitting** the form data to a **form-handler**.

The form-handler is typically a page on the server with a script for processing input data.

The form-handler is specified in the form's **action** attribute.

Example

A form with a submit button:

```
<form action="/action_page.php">
                    <label for="fname">First name:</label><br>
                    <input type="text" id="fname" name="fname" value="John
                  "><br>
                    <label for="lname">Last name:</label><br>
                    <input type="text" id="lname" name="lname" value="Doe"
                  ><br><br>
                    <input type="submit" value="Submit">
                  </form>
```

The Action Attribute

The `action` attribute defines the action to be performed when the form is submitted.

Usually, the form data is sent to a page on the server when the user clicks on the submit button.

In the example above, the form data is sent to a page on the server called "/action_page.php". This page contains a server-side script that handles the form data:

```
<form action="/action_page.php">
```

If the `action` attribute is omitted, the action is set to the current page.

The Target Attribute

The `target` attribute specifies if the submitted result will open in a new browser tab, a frame, or in the current window.

The default value is "`_self`" which means the form will be submitted in the current window.

To make the form result open in a new browser tab, use the value "`_blank`".

Example

Here, the submitted result will open in a new browser tab:

```
<form action="/action_page.php" target="_blank">
```

HTML Input Types

HTML Input Types

Here are the different input types you can use in HTML:

- `<input type="button">`
- `<input type="checkbox">`
- `<input type="color">`
- `<input type="date">`
- `<input type="datetime-local">`
- `<input type="email">`
- `<input type="file">`
- `<input type="hidden">`
- `<input type="image">`
- `<input type="month">`
- `<input type="number">`
- `<input type="password">`
- `<input type="radio">`
- `<input type="range">`
- `<input type="reset">`
- `<input type="search">`
- `<input type="submit">`
- `<input type="tel">`
- `<input type="text">`
- `<input type="time">`
- `<input type="url">`
- `<input type="week">`

Input Type Password

`<input type="password">` defines a **password field**:

Example

```
<form>
  <label for="username">Username:</label><br>
  <input type="text" id="username" name="username"><br>
  <label for="pwd">Password:</label><br>
  <input type="password" id="pwd" name="pwd">
</form>
```

Input Type Submit

`<input type="submit">` defines a button for **submitting** form data to a **form-handler**.

The form-handler is typically a server page with a script for processing input data.

The form-handler is specified in the form's `action` attribute:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

Input Type Reset

`<input type="reset">` defines a **reset button** that will reset all form values to their default values:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
```

```
  <input type="submit" value="Submit">
  <input type="reset">
</form>
```

Input Type Radio

`<input type="radio">` defines a **radio button**.

Radio buttons let a user select ONLY ONE of a limited number of choices:

Example

```
<form>
  <input type="radio" id="male" name="gender" value="male">
  <label for="male">Male</label><br>
  <input type="radio" id="female" name="gender" value="female">
  <label for="female">Female</label><br>
  <input type="radio" id="other" name="gender" value="other">
  <label for="other">Other</label>
</form>
```

Input Type Checkbox

`<input type="checkbox">` defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

Example

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```

Input Type Button

`<input type="button">` defines a **button**:

Example

```
<input type="button" onclick="alert('Hello World!')" value="Click Me!">
```

Input Type Color

The `<input type="color">` is used for input fields that should contain a color.

Depending on browser support, a color picker can show up in the input field.

Example

```
<form>
  <label for="favcolor">Select your favorite color:</label>
  <input type="color" id="favcolor" name="favcolor">
</form>
```

HTML Tables

Defining an HTML Table

An HTML table is defined with the `<table>` tag.

Each table row is defined with the `<tr>` tag. A table header is defined with the `<th>` tag. By default, table headings are bold and centered. A table data/cell is defined with the `<td>` tag.

Example

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
```

```
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Note: The `<td>` elements are the data containers of the table. They can contain all sorts of HTML elements; text, images, lists, other tables, etc.

HTML Table - Adding a Border

If you do not specify a border for the table, it will be displayed without borders.

A border is set using the CSS `border` property:

Example

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

HTML Table - Collapsed Borders

If you want the borders to collapse into one border, add the CSS `border-collapse` property:

Example

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

HTML Table - Adding Cell Padding

Cell padding specifies the space between the cell content and its borders.

If you do not specify a padding, the table cells will be displayed without padding.

To set the padding, use the CSS `padding` property:

Example

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

HTML Table - Left-align Headings

By default, table headings are bold and centered.

To left-align the table headings, use the CSS `text-align` property:

Example

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

HTML Table - Adding Border Spacing

Border spacing specifies the space between the cells.

To set the border spacing for a table, use the CSS `border-spacing` property:

Example

```
table {
  border-spacing: 5px;
}
```

HTML Table - Cells that Span Many Columns

To make a cell span more than one column, use the `colspan` attribute:

Example

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th colspan="2">Telephone</th>
  </tr>
  <tr>
    <td>Bill Gates</td>
    <td>55577854</td>
    <td>55577855</td>
  </tr>
</table>
```

A Special Style for One Table

To define a special style for a special table, add an `id` attribute to the table:

Example

```
<table id="t01">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

JSON - Introduction

JSON: **J**ava**S**cript **O**bject **N**otation.

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

Exchanging Data

When exchanging data between a browser and a server, the data can only be text.

JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

We can also convert any JSON received from the server into JavaScript objects.

This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

Sending Data

If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

Example

```
var myObj = {name: "John", age: 31, city: "New York"};
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

You will learn more about the `JSON.stringify()` function later in this tutorial.

Receiving Data

If you receive data in JSON format, you can convert it into a JavaScript object:

Example

```
var myJSON = '{"name":"John", "age":31, "city":"New York"}';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

Storing Data

When storing data, the data has to be a certain format, and regardless of where you choose to store it, *text* is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

Example

Storing data in local storage

```
// Storing data:
myObj = {name: "John", age: 31, city: "New York"};
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

// Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent *

*
JSON uses JavaScript syntax, but the JSON format is text only.
Text can be read and used as a data format by any programming language.

Why use JSON?

Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

`JSON.parse()`

So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

JSON Syntax

The JSON syntax is a subset of the JavaScript syntax.

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data - A Name and a Value

JSON data is written as name/value pairs.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

Example

```
"name":"John"
```

JSON names require double quotes. JavaScript names don't.

JSON - Evaluates to JavaScript Objects

The JSON format is almost identical to JavaScript objects.

In JSON, *keys* must be strings, written with double quotes:

JSON

```
{ "name":"John" }
```

In JavaScript, keys can be strings, numbers, or identifier names:

JavaScript

```
{ name:"John" }
```

JSON Values

In **JSON**, *values* must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

In **JavaScript** values can be all of the above, plus any other valid JavaScript expression, including:

- a function
- a date
- undefined

In JSON, *string values* must be written with double quotes:

JSON

```
{ "name":"John" }
```

In JavaScript, you can write string values with double *or* single quotes:

JavaScript

```
{ name:'John' }
```

JSON Uses JavaScript Syntax

Because JSON syntax is derived from JavaScript object notation, very little extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an object and assign data to it, like this:

Example

```
var person = { name: "John", age: 31, city: "New York" };
```

You can access a JavaScript object like this:

Example

```
// returns John
person.name;
```

JavaScript Arrays as JSON

The same way JavaScript objects can be used as JSON, JavaScript arrays can also be used as JSON.

You will learn more about arrays as JSON later in this tutorial.

JSON Files

- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json"

Exercise:

Add a "tooltip" to the paragraph below with the text "About MySchools".

```
<p ="About MySchools">W3Schools is a web developer's site.</p>
```

Exercise:

Set the size of the image to 250 pixels wide and 400 pixels tall.

```
<img src="Myschools.jpg" width=" " height=" ">
```

Exercise:

Make the element below into a link that goes to "https://www.Myschools.com".

```
<a "https://www.Myschools.com">This is a link</a>
```

Exercise:

Specify an alternate text for the image.

Alternate text is useful when the image cannot be displayed, like when the page is read by a screen reader.

```
<img src="Myschools.png" [ ]="Myschools Logo">
```

Exercise:

Use CSS to set the background color of the document (body) to yellow.

```
<!DOCTYPE html>
<html>
<head>
<style>
[ ] [ ]
  [ ]:yellow;
[ ]
</style>
</head>
<body>

<h1>My Home Page</h1>

</body>
</html>
```

Exercise:

Use CSS to set the font of the document to "courier".

```
<!DOCTYPE html>
<html>
<head>
<style>
body {        :courier;}
</style>
</head>
<body>

<h1>My Home Page</h1>

</body>
</html>
```

Exercise:

Use CSS to set the text color of the document to red.

```
<!DOCTYPE html>
<html>
<head>
<style>
body {        :red;}
</style>
</head>
<body>

<h1>My Home Page</h1>

</body>
</html>
```

Exercise:

Use CSS to make a yellow, 1 pixel thick, border around all paragraphs.

```
<!DOCTYPE html>
<html>
<head>
<style>
[ ] {[ ]: [ ] solid [ ];}
</style>
</head>
<body>

<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>

</body>
</html>
```

Exercise:

Use the correct attribute to make sure that the last paragraph gets the styling as described in the `style` element.

```
<!DOCTYPE html>
<html>
<head>
<style>
#special {
  color:gray;
  background-color:lightblue;
```

```
}
</style>
</head>
<body>

<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p >This is a paragraph.</p>

</body>
</html>
```

Exercise:

In the form below, add an empty drop down list with the name "cars".

```
<form action="/action_page.php">
<>
</>
</form>
```

Exercise:

In the form below, add a text area with the name "note".

```
<form action="/action_page.php">
<></>
</form>
```

Exercise:

In the form below, add an input field for text, with the name "username" .

```
<form action="/action_page.php">
<      >
</form>
```

Exercise:

1. In the form below, add a input field that can only contain numbers
2. Use the correct input attributes to only allow numbers between 1 and 5.

```
<form action="/action_page.php">
<input type="      " "      ="1" "      ="5">
</form>
```