

HTML5 Tutorial

With HTML you can create your own Website.
This tutorial teaches you everything about HTML.
HTML is easy to learn - You will enjoy it.

Examples in Every Chapter

This HTML tutorial contains hundreds of HTML examples.
With our online HTML editor, you can edit the HTML, and click on a button to view the result.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

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

</body>
</html>
```

Click on the "Try it Yourself" button to see how it works.

[Start learning HTML now!](#)

HTML Examples

At the end of the HTML tutorial, you can find more than 200 examples.

With our online editor, you can edit and test each example yourself.

[Go to HTML Examples!](#)

HTML Exercises and Quiz Test

Test your HTML skills at W3Schools!

[Start HTML Exercises!](#)

[Start HTML Quiz!](#)

HTML References

At W3Schools you will find complete references about tags, attributes, events, color names, entities, character-sets, URL encoding, language codes, HTTP messages, and more.

[HTML Tag Reference](#)

HTML Introduction

What is HTML?

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

A Simple HTML Document

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

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

</body>
</html>
```

Example Explained

- The `<!DOCTYPE html>` declaration defines this document to be HTML5
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the document
- The `<title>` element specifies a title for the document
- The `<body>` element contains the visible page content
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

HTML Tags

HTML tags are element names surrounded by angle brackets:

```
<tagname>content goes here...</tagname>
```

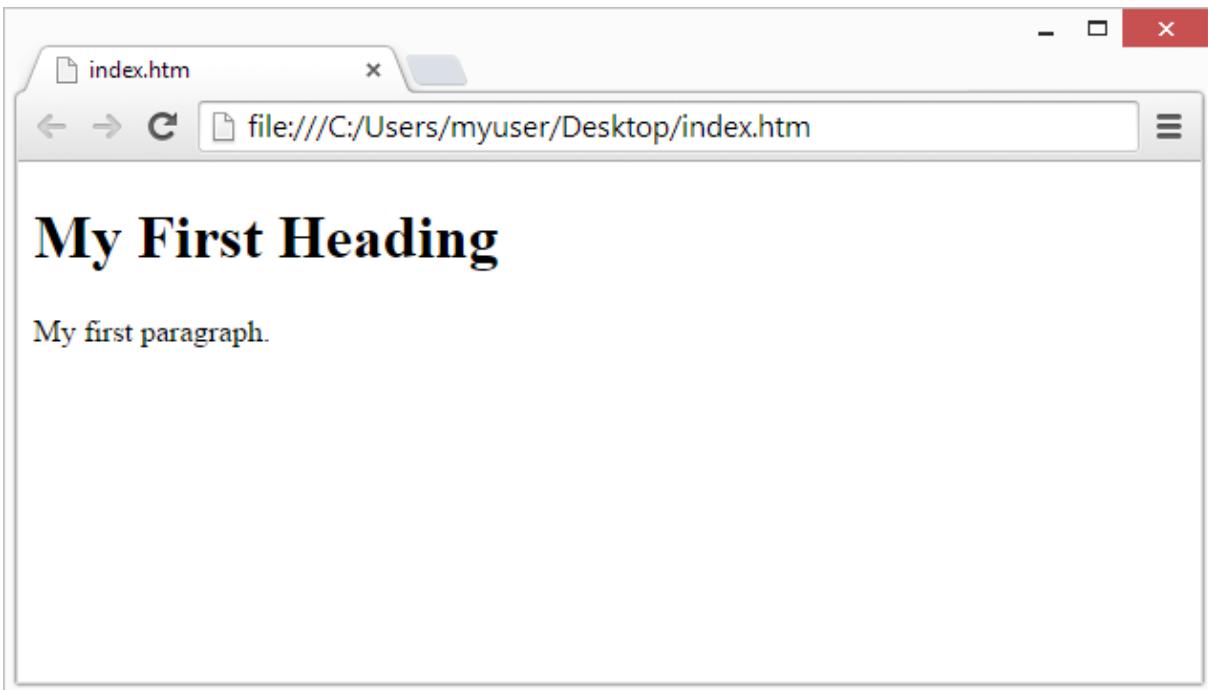
- HTML tags normally come **in pairs** like `<p>` and `</p>`
- The first tag in a pair is the **start tag**, the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

Tip: The start tag is also called the **opening tag**, and the end tag the **closing tag**.

Web Browsers

The purpose of a web browser (Chrome, IE, Firefox, Safari) is to read HTML documents and display them.

The browser does not display the HTML tags, but uses them to determine how to display the document:



HTML Page Structure

Below is a visualization of an HTML page structure:

```
<html>
```

```
  <head>
```

```
    <title>Page title</title>
```

```
  </head>
```

```
<body>
```

```
  <h1>This is a heading</h1>
```

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

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

```
</body>
```

```
</html>
```

Note: Only the content inside the `<body>` section (the white area above) is displayed in a browser.

The `<!DOCTYPE>` Declaration

The `<!DOCTYPE>` declaration represents the document type, and helps browsers to display web pages correctly.

It must only appear once, at the top of the page (before any HTML tags).

The `<!DOCTYPE>` declaration is not case sensitive.

The `<!DOCTYPE>` declaration for HTML5 is:

```
<!DOCTYPE html>
```

HTML Versions

Since the early days of the web, there have been many versions of HTML:

Version	Year
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

HTML Editors

Write HTML Using Notepad or TextEdit

Web pages can be created and modified by using professional HTML editors.

However, for learning HTML we recommend a simple text editor like Notepad (PC) or TextEdit (Mac).

We believe using a simple text editor is a good way to learn HTML.

Follow the four steps below to create your first web page with Notepad or TextEdit.

Step 1: Open Notepad (PC)

Windows 8 or later:

Open the **Start Screen** (the window symbol at the bottom left on your screen). Type **Notepad**.

Windows 7 or earlier:

Open **Start > Programs > Accessories > Notepad**

Step 1: Open TextEdit (Mac)

Open **Finder > Applications >TextEdit**

Also change some preferences to get the application to save files correctly. In **Preferences > Format >** choose "**Plain Text**"

Then under "Open and Save", check the box that says "Display HTML files as HTML code instead of formatted text".

Then open a new document to place the code.

Step 2: Write Some HTML

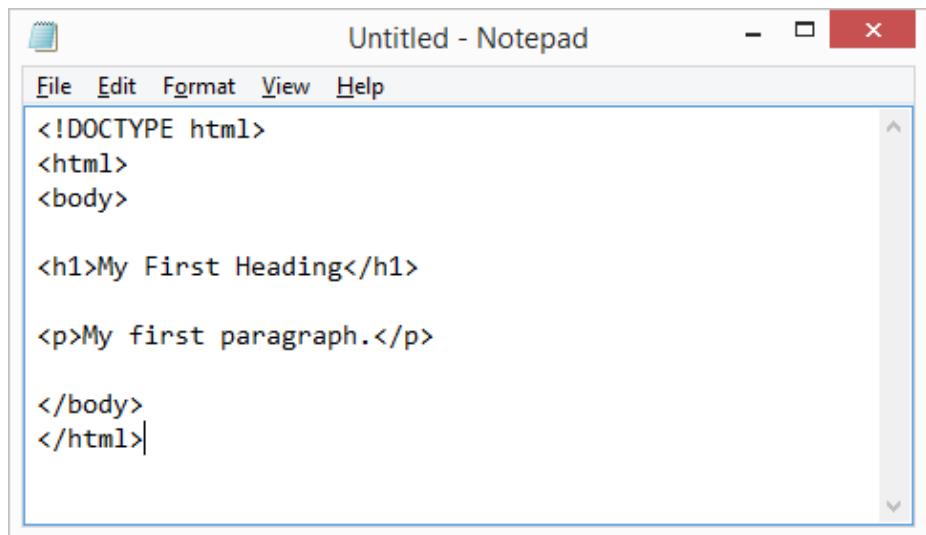
Write or copy some HTML into Notepad.

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

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```



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

<h1>My First Heading</h1>

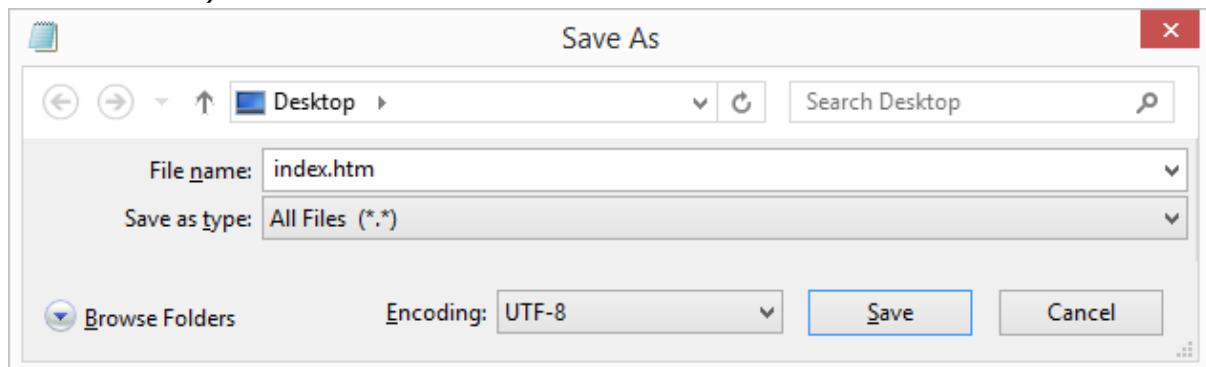
<p>My first paragraph.</p>

</body>
</html>
```

Step 3: Save the HTML Page

Save the file on your computer. Select **File > Save as** in the Notepad menu.

Name the file "**index.htm**" and set the encoding to **UTF-8** (which is the preferred encoding for HTML files).

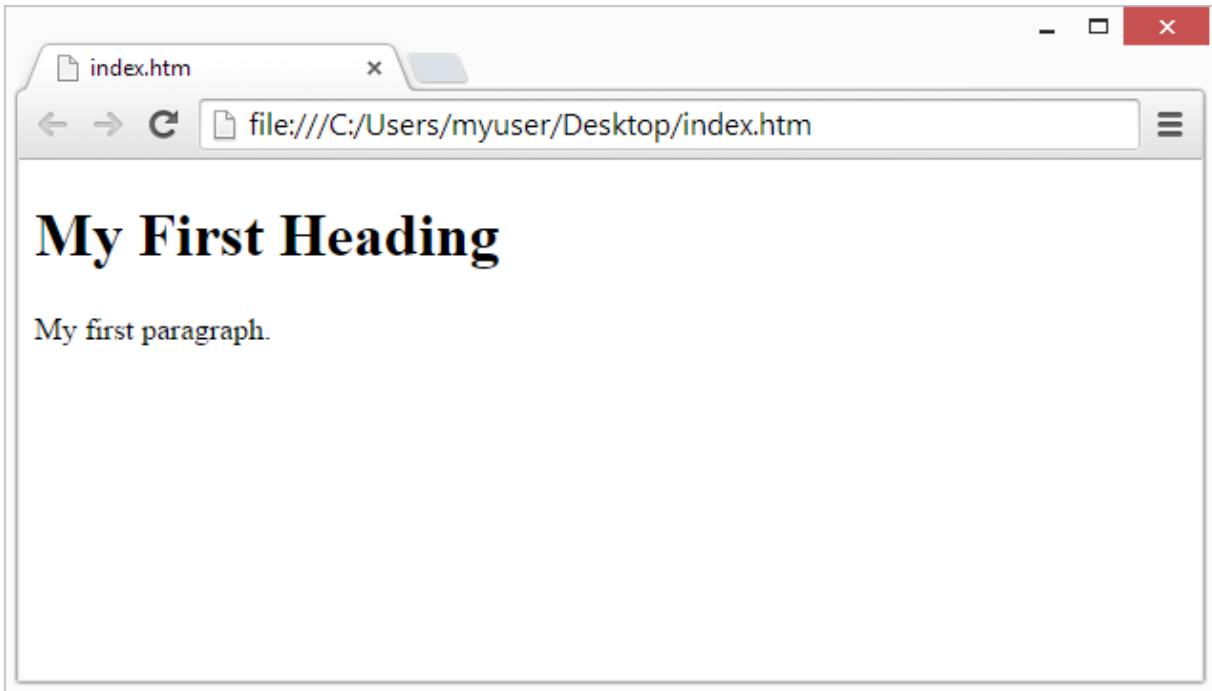


You can use either .htm or .html as file extension. There is no difference, it is up to you.

Step 4: View the HTML Page in Your Browser

Open the saved HTML file in your favorite browser (double click on the file, or right-click - and choose "Open with").

The result will look much like this:



W3Schools Online Editor

With our free online editor, you can edit HTML code and view the result in your browser. It is the perfect tool when you want to **test** code fast. It also has color coding and the ability to save and share code with others:

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

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

</body>
</html>
```

Click on the "Try it Yourself" button to see how it works.

HTML Basic Examples

Don't worry if these examples use tags you have not learned.

You will learn about them in the next chapters.

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>`.

Example

```
<!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 `` tag.

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

Example

```

```

HTML Buttons

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

Example

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

HTML Lists

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

Example

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

HTML Elements

HTML Elements

An HTML element usually consists of a **start** tag and **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:

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

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

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>`.

The element **content** is another HTML element (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>`.

The element **content** 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 `<p>` and an **end** tag `</p>`.

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.

`
` is an empty element without a closing tag (the `
` tag defines a line break).

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.

Use Lowercase Tags

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.

At W3Schools we always use lowercase tags.

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.w3schools.com">This is a link</a>
```

You will learn more about links and the `<a>` tag later in this tutorial.

The src Attribute

HTML images are defined with the `` tag.

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

Example

```

```

The width and height Attributes

Images in HTML have a set of **size** attributes, which specifies the width and height of the image:

Example

```

```

The image size is specified in pixels: `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, when an image cannot be displayed.

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

Example

```

```

The `alt` attribute is also useful if the image does not exist:

Example

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

```

```

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">I am a paragraph</p>
```

You will learn more about styling later in this tutorial, and in our [CSS Tutorial](#).

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, use 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>
```

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.

At W3Schools we always use lowercase attribute names.

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:

Bad

```
<a href=https://www.w3schools.com>
```

Good

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

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

Sometimes it is **necessary** to use quotes. This example will not display the title attribute correctly, because it contains a space:

Example

```
<p title=About W3Schools>
```

Using quotes are the most common. Omitting quotes can produce errors. At W3Schools we **always** use quotes around attribute values.

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
- At W3Schools we always use **lowercase** attribute names
- At W3Schools we always **quote** attribute values with double quotes

HTML Attributes

Below is an alphabetical list of some attributes often used in HTML, which you will learn more about in this tutorial:

Attribute	Description
alt	Specifies an alternative text for an image, when the image cannot be displayed
disabled	Specifies that an input element should be disabled
href	Specifies the URL (web address) for a link
id	Specifies a unique id for an element
src	Specifies the URL (web address) for an image
style	Specifies an inline CSS style for an element
title	Specifies extra information about an element (displayed as a tool tip)

A complete list of all attributes for each HTML element, is listed in our: [HTML Attribute Reference](#).

HTML Headings

Headings

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

HTML Headings

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

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

Example

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

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 skim your pages 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:

Example

```
<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 has nothing to do with HTML headings.

The `<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:

Example

```
<!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, links, 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 IE), 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 Tag Reference

W3Schools' tag reference contains additional information about these tags and their attributes. You will learn more about HTML tags and attributes in the next chapters of this tutorial.

Tag	Description
<u><html></u>	Defines the root of an HTML document
<u><body></u>	Defines the document's body
<u><head></u>	A container for all the head elements (title, scripts, styles, meta information, and more)
<u><h1> to <h6></u>	Defines HTML headings
<u><hr></u>	Defines a thematic change in the content

HTML Paragraphs

HTML Paragraphs

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

Example

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

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

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:

Example

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

The example above will work in most browsers, but do not rely on it.

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

HTML Line Breaks

The HTML `
` element defines a **line break**.

Use `
` 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 `
` tag is an empty tag, which means that it has no end tag.

The Poem Problem

This poem will display on a single line:

Example

```
<p>
    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.
</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:

Example

```
<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 Tag Reference

W3Schools' tag reference contains additional information about HTML elements and their attributes.

Tag	Description
<u><p></u>	Defines a paragraph
<u>
</u>	Inserts a single line break
<u><pre></u>	Defines pre-formatted text

HTML Styles

Example

```
I am Red
```

```
I am Blue
```

```
I am Big
```

The HTML Style Attribute

Setting the style of an HTML element, can be done with the `style` attribute.

The HTML `style` attribute has the following **syntax**:

```
<tagname style="property:value;">
```

The **property** is a CSS property. The **value** is a CSS value.

You will learn more about CSS later in this tutorial.

HTML Background Color

The `background-color` property defines the background color for an HTML element.

This example sets the background color for a page to powderblue:

Example

```
<body style="background-color:powderblue;">

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

</body>
```

HTML Text Color

The `color` property defines the text color for an HTML element:

Example

```
<h1 style="color:blue;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>
```

HTML Fonts

The `font-family` property defines the font to be used for an HTML element:

Example

```
<h1 style="font-family:verdana;">This is a heading</h1>
<p style="font-family:courier;">This is a paragraph.</p>
```

HTML Text Size

The `font-size` property defines the text size for an HTML element:

Example

```
<h1 style="font-size:300%;">This is a heading</h1>
<p style="font-size:160%;">This is a paragraph.</p>
```

HTML Text Alignment

The `text-align` property defines the horizontal text alignment for an HTML element:

Example

```
<h1 style="text-align:center;">Centered Heading</h1>
<p style="text-align:center;">Centered paragraph.</p>
```

Chapter Summary

- Use the `style` attribute for styling HTML elements
- Use `background-color` for background color
- Use `color` for text colors
- Use `font-family` for text fonts
- Use `font-size` for text sizes
- Use `text-align` for text alignment

HTML Text Formatting

Text Formatting

This text is bold

This text is italic

This is _{subscript} and ^{superscript}

HTML Formatting Elements

In the previous chapter, you learned about the HTML **style attribute**.

HTML also defines special **elements** for defining text with a special **meaning**.

HTML uses elements like `` and `<i>` for formatting output, like **bold** or *italic* text.

Formatting elements were designed to display special types of text:

- `` - Bold text
- `` - Important text
- `<i>` - Italic text
- `` - Emphasized text
- `<mark>` - Marked text
- `<small>` - Small text
- `` - Deleted text
- `<ins>` - Inserted text
- `<sub>` - Subscript text
- `<sup>` - Superscript text

HTML `` and `` Elements

The HTML `` element defines **bold** text, without any extra importance.

Example

```
<b>This text is bold</b>
```

The HTML `` element defines **strong** text, with added semantic "strong" importance.

Example

```
<strong>This text is strong</strong>
```

HTML *<i>* and ** Elements

The HTML `<i>` element defines *italic* text, without any extra importance.

Example

```
<i>This text is italic</i>
```

The HTML `` element defines *emphasized* text, with added semantic importance.

Example

```
<em>This text is emphasized</em>
```

Note: Browsers display `` as ``, and `` as `<i>`. However, there is a difference in the meaning of these tags: `` and `<i>` defines bold and italic text, but `` and `` means that the text is "important".

HTML *<small>* Element

The HTML `<small>` element defines smaller text:

Example

```
<h2>HTML <small>Small</small> Formatting</h2>
```

HTML <mark> Element

The HTML `<mark>` element defines marked or highlighted text:

Example

```
<h2>HTML <mark>Marked</mark> Formatting</h2>
```

HTML Element

The HTML `` element defines deleted (removed) text.

Example

```
<p>My favorite color is <del>blue</del> red.</p>
```

HTML <ins> Element

The HTML `<ins>` element defines inserted (added) text.

Example

```
<p>My favorite <ins>color</ins> is red.</p>
```

HTML <sub> Element

The HTML `<sub>` element defines subscripted text.

Example

```
<p>This is <sub>subscripted</sub> text.</p>
```

HTML <sup> Element

The HTML `<sup>` element defines superscripted text.

Example

```
| <p>This is <sup>superscripted</sup> text.</p>
```

HTML Text Formatting Elements

Tag	Description
<code></code>	Defines bold text
<code></code>	Defines emphasized text
<code><i></code>	Defines italic text
<code><small></code>	Defines smaller text
<code></code>	Defines important text
<code><sub></code>	Defines subscripted text
<code><sup></code>	Defines superscripted text
<code><ins></code>	Defines inserted text
<code></code>	Defines deleted text
<code><mark></code>	Defines marked/highlighted text

HTML Quotation and Citation Elements

Quotation

Here is a quote from WWF's website:

For 50 years, WWF has been protecting the future of nature. The world's leading conservation organization, WWF works in 100 countries and is supported by 1.2 million members in the United States and close to 5 million globally.

HTML <q> for Short Quotations

The HTML <q> element defines a short quotation.

Browsers usually insert quotation marks around the <q> element.

Example

```
<p>WWF's goal is to: <q>Build a future where people live in harmony with  
nature.</q></p>
```

HTML <blockquote> for Quotations

The HTML <blockquote> element defines a section that is quoted from another source.

Browsers usually indent <blockquote> elements.

Example

```
<p>Here is a quote from WWF's website:</p>  
<blockquote cite="http://www.worldwildlife.org/who/index.html">  
For 50 years, WWF has been protecting the future of nature.  
The world's leading conservation organization,  
WWF works in 100 countries and is supported by  
1.2 million members in the United States and  
close to 5 million globally.  
</blockquote>
```

HTML <abbr> for Abbreviations

The HTML `<abbr>` element defines an abbreviation or an acronym.

Marking abbreviations can give useful information to browsers, translation systems and search-engines.

Example

```
<p>The <abbr title="World Health Organization">WHO</abbr> was founded in  
1948.</p>
```

HTML <address> for Contact Information

The HTML `<address>` element defines contact information (author/owner) of a document or an article.

The `<address>` element is usually displayed in italic. Most browsers will add a line break before and after the element.

Example

```
<address>  
Written by John Doe.<br>  
Visit us at:<br>  
Example.com<br>  
Box 564, Disneyland<br>  
USA  
</address>
```

HTML <cite> for Work Title

The HTML `<cite>` element defines the title of a work.

Browsers usually display `<cite>` elements in italic.

Example

```
<p><cite>The Scream</cite> by Edvard Munch. Painted in 1893.</p>
```

HTML <bdo> for Bi-Directional Override

The HTML `<bdo>` element defines bi-directional override.

The `<bdo>` element is used to override the current text direction:

Example

```
<bdo dir="rtl">This text will be written from right to left</bdo>
```

HTML Quotation and Citation Elements

Tag	Description
<code><abbr></code>	Defines an abbreviation or acronym
<code><address></code>	Defines contact information for the author/owner of a document
<code><bdo></code>	Defines the text direction
<code><blockquote></code>	Defines a section that is quoted from another source
<code><cite></code>	Defines the title of a work
<code><q></code>	Defines a short inline quotation

HTML Comments

Comment tags are used to insert comments in the HTML source code.

HTML Comment Tags

You can add comments to your HTML source by using the following syntax:

```
<!-- Write your comments here -->
```

Notice that there is an exclamation point (!) in the opening tag, but not in the closing tag.

Note: Comments are not displayed by the browser, but they can help document your HTML source code.

With comments you can place notifications and reminders in your HTML:

Example

```
<!-- This is a comment -->  
  
<p>This is a paragraph.</p>  
  
<!-- Remember to add more information here -->
```

Comments are also great for debugging HTML, because you can comment out HTML lines of code, one at a time, to search for errors:

Example

```
<!-- Do not display this at the moment  
  
-->
```

HTML Colors

HTML colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

Color Names

In HTML, a color can be specified by using a color name:

Tomato

Orange

DodgerBlue

MediumSeaGreen

Gray

SlateBlue

Violet

LightGray

[Try it Yourself »](#)

HTML supports [140 standard color names](#).

Background Color

You can set the background color for HTML elements:

Hello World

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Example

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

Text Color

You can set the color of text:

Hello World

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

 Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Example

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Lorem ipsum...</p>
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

Border Color

You can set the color of borders:

Hello World

Hello World

Hello World

Example

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>
```

Color Values

In HTML, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

Same as color name "Tomato":

```
rgb(255, 99, 71)
```

```
#ff6347
```

```
hsl(9, 100%, 64%)
```

Same as color name "Tomato", but 50% transparent:

```
rgba(255, 99, 71, 0.5)
```

```
hsla(9, 100%, 64%, 0.5)
```

Example

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>
```

RGB Value

In HTML, a color can be specified as an RGB value, using this formula:

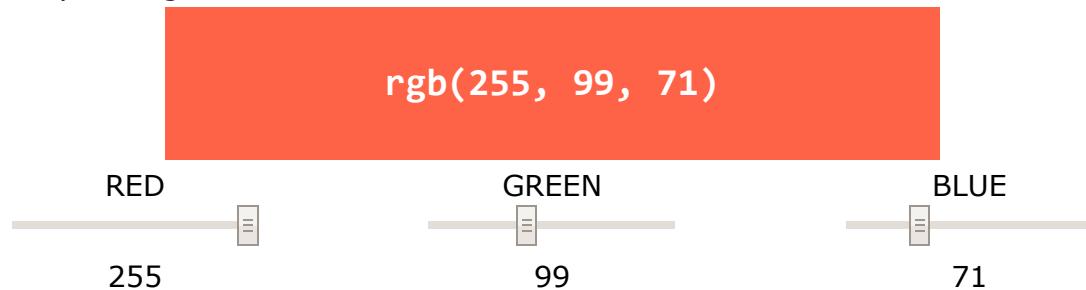
rgb(*red*, *green*, *blue*)

Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255. For example, `rgb(255, 0, 0)` is displayed as red, because red is set to its highest value (255) and the others are set to 0.

To display the color black, all color parameters must be set to 0, like this: `rgb(0, 0, 0)`.

To display the color white, all color parameters must be set to 255, like this: `rgb(255, 255, 255)`.

Experiment by mixing the RGB values below:



Example

`rgb(255, 0, 0)`

`rgb(0, 0, 255)`

`rgb(60, 179, 113)`

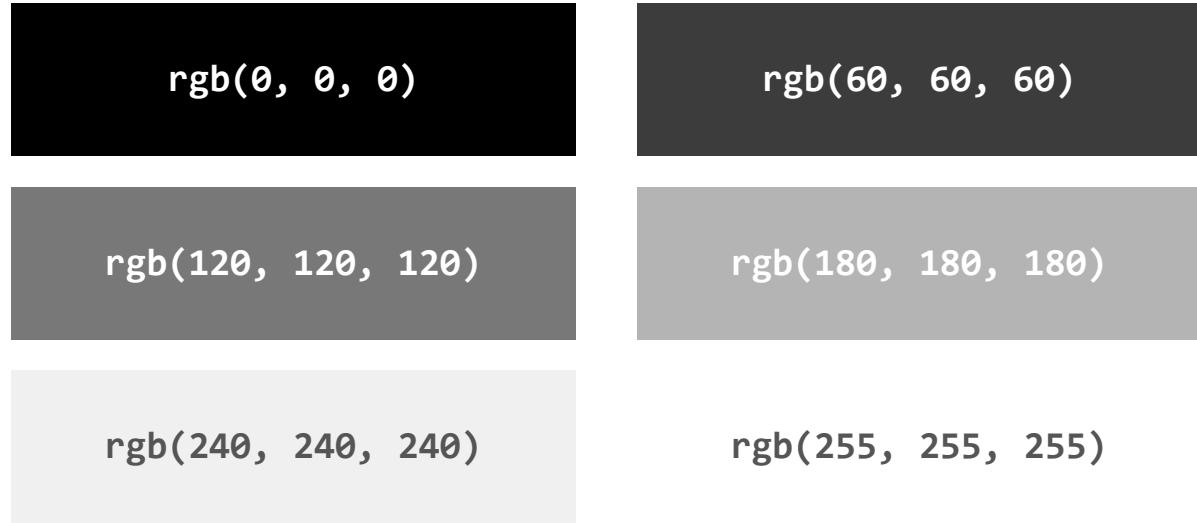
`rgb(238, 130, 238)`

`rgb(255, 165, 0)`

`rgb(106, 90, 205)`

Shades of gray are often defined using equal values for all the 3 light sources:

Example



HEX Value

In HTML, a color can be specified using a hexadecimal value in the form:

#rrggbb

Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).

For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).

Example



Shades of gray are often defined using equal values for all the 3 light sources:

Example

#000000

#3c3c3c

#787878

#b4b4b4

#f0f0f0

#ffffff

HSL Value

In HTML, a color can be specified using hue, saturation, and lightness (HSL) in the form:

hsl(*hue*, *saturation*, *lightness*)

Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
Saturation is a percentage value, 0% means a shade of gray, and 100% is the full color.
Lightness is also a percentage, 0% is black, 50% is neither light or dark, 100% is white

Example

hsl(0, 100%, 50%)

hsl(240, 100%, 50%)

hsl(147, 50%, 47%)

hsl(300, 76%, 72%)

hsl(39, 100%, 50%)

hsl(248, 53%, 58%)

Saturation

Saturation can be described as the intensity of a color.

100% is pure color, no shades of gray

50% is 50% gray, but you can still see the color.

0% is completely gray, you can no longer see the color.

Example

`hsl(0, 100%, 50%)`

`hsl(0, 80%, 50%)`

`hsl(0, 60%, 50%)`

`hsl(0, 40%, 50%)`

`hsl(0, 20%, 50%)`

`hsl(0, 0%, 50%)`

Lightness

The lightness of a color can be described as how much light you want to give the color, where 0% means no light (black), 50% means 50% light (neither dark nor light) 100% means full lightness (white).

Example

`hsl(0, 100%, 0%)`

`hsl(0, 100%, 25%)`

`hsl(0, 100%, 50%)`

`hsl(0, 100%, 75%)`

`hsl(0, 100%, 90%)`

`hsl(0, 100%, 100%)`

Shades of gray are often defined by setting the hue and saturation to 0, and adjust the lightness from 0% to 100% to get darker/lighter shades:

Example

`hsl(0, 0%, 0%)`

`hsl(0, 0%, 24%)`

`hsl(0, 0%, 47%)`

`hsl(0, 0%, 71%)`

`hsl(0, 0%, 94%)`

`hsl(0, 0%, 100%)`

RGBA Value

RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color.

An RGBA color value is specified with:

`rgba(red, green, blue, alpha)`

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

Example

`rgba(255, 99, 71, 0)`

`rgba(255, 99, 71, 0.2)`

`rgba(255, 99, 71, 0.4)`

`rgba(255, 99, 71, 0.6)`

`rgba(255, 99, 71, 0.8)`

`rgba(255, 99, 71, 1)`

HSLA Value

HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color.

An HSLA color value is specified with:

hsla(*hue*, *saturation*, *lightness*, *alpha*)

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

Example

`hsla(9, 100%, 64%, 0)`

`hsla(9, 100%, 64%, 0.2)`

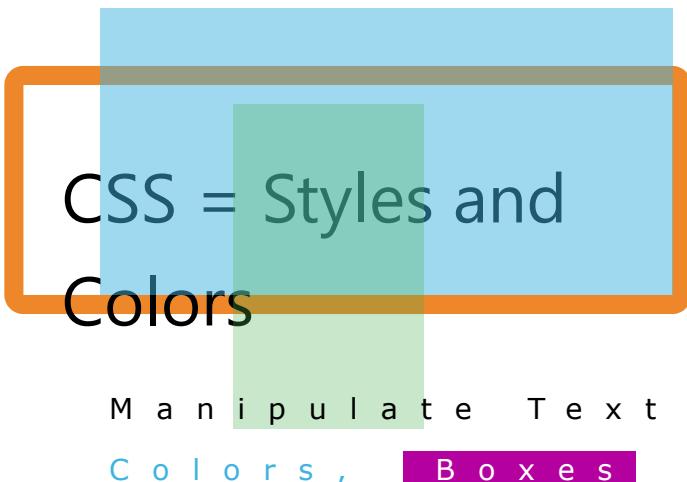
`hsla(9, 100%, 64%, 0.4)`

`hsla(9, 100%, 64%, 0.6)`

`hsla(9, 100%, 64%, 0.8)`

`hsla(9, 100%, 64%, 1)`

HTML Styles - CSS



Styling HTML with CSS

CSS stands for **Cascading Style Sheets**.

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.

Example

```
<!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:

Example

```
p {  
    border: 1px solid powderblue;  
    padding: 30px;  
}
```

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;  
}
```

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

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

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

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

Example

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

This example links to a style sheet located in the same folder as the current page:

Example

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

You can read more about file paths in the chapter [HTML File Paths](#).

Chapter Summary

- Use the HTML `style` attribute for inline styling
- Use the HTML `<style>` element to define internal CSS
- Use the HTML `<link>` element to refer to an external CSS file
- Use the HTML `<head>` element to store `<style>` and `<link>` elements
- Use the CSS `color` property for text colors
- Use the CSS `font-family` property for text fonts
- Use the CSS `font-size` property for text sizes
- Use the CSS `border` property for borders
- Use the CSS `padding` property for space inside the border
- Use the CSS `margin` property for space outside the border

HTML Style Tags

Tag	Description
<code><style></code>	Defines style information for an HTML document
<code><link></code>	Defines a link between a document and an external resource

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

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

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

Example

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

The `href` attribute specifies the destination address (<https://www.w3schools.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 http://www....).

Example

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

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>
```

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.w3schools.com/" target="_blank">Visit W3Schools!</a>
```

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

Example

```
<a href="https://www.w3schools.com/html/" target="_top">HTML5 tutorial!</a>
```

HTML Links - Image as Link

It is common to use images as links:

Example

```
<a href="default.asp">
  
</a>
```

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

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.w3schools.com/html/" title="Go to W3Schools HTML section">Visit our HTML Tutorial</a>
```

HTML Links - Create a Bookmark

HTML bookmarks are used to allow readers to jump to specific parts of a Web page. Bookmarks can be useful if your webpage is very long.

To make a bookmark, you must first create the bookmark, and then add a link to it. When the link is clicked, the page will scroll 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:

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

Or, add a link to the bookmark ("Jump to Chapter 4"), from another page:

Example

```
<a href="html_demo.html#C4">Jump to Chapter 4</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.w3schools.com/html/default.asp">HTML tutorial</a>
```

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

Example

```
<a href="/html/default.asp">HTML tutorial</a>
```

This example links to a page located in the same folder as the current page:

Example

```
<a href="default.asp">HTML tutorial</a>
```

You can read more about file paths in the chapter [HTML File Paths](#).

Chapter Summary

- Use the `<a>` element to define a link
- Use the `href` attribute to define the link address
- Use the `target` attribute to define where to open the linked document
- Use the `` element (inside `<a>`) to use an image as a link
- Use the `id` attribute (`id="value"`) to define bookmarks in a page
- Use the `href` attribute (`href="#value"`) to link to the bookmark

HTML Link Tags

Tag	Description
<u><a></u>	Defines a hyperlink

HTML Images

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



Example

```

```

Example

```

```

Example

```

```

HTML Images Syntax

In HTML, images are defined with the `` tag.

The `` 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:

```

```

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:

Example

```

```

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

Example

```

```

Note: The `alt` attribute is required. A web page will not validate correctly without it.

Image Size - Width and Height

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

Example

```

```

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

Example

```

```

The `width` and `height` attributes always defines the width and height of the image in pixels.

Note: Always specify the width and height of an image. If width and height are not specified, the page might flicker while the image loads.

Width and Height, or Style?

The `width`, `height`, and `style` attributes are valid in HTML5.

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

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
    width: 100%;
}
</style>
</head>
<body>




</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

```

```

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

```

```

You can read more about file paths in the chapter [HTML File Paths](#).

Animated Images

HTML allows animated GIFs:

Example

```

```

Image as a Link

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

Example

```
<a href="default.asp">
  
</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>
The image will float to the right of the text.</p>

<p>
The image will float to the left of the text.</p>
```

Tip: To learn more about CSS Float, read our [CSS Float Tutorial](#).

Image Maps

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

In the image below, click on the computer, the phone, or the cup of coffee:



Example

```


<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>
```

The `name` attribute of the `<map>` tag is associated with the ``'s `usemap` attribute and creates a relationship between the image and the map.

The `<map>` element contains a number of `<area>` tags, that define the clickable areas in the image-map.

Background Image

To add a background image on an HTML element, use the CSS property `background-image`:

Example

To add a background image on a web page, specify the `background-image` property on the `BODY` element:

```
<body style="background-image:url('clouds.jpg')">  
  
<h2>Background Image</h2>  
  
</body>
```

Example

To add a background image on a paragraph, specify the `background-image` property on the `P` element:

```
<body>  
  
<p style="background-image:url('clouds.jpg')">  
...  
</p>  
  
</body>
```

To learn more about background images, study our [CSS Background Tutorial](#).

The <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.

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

Example

Show one picture if the browser window (viewport) is a minimum of 650 pixels, and another image if not, but larger than 465 pixels.

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

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

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 blind, visually impaired, or learning disabled.

Chapter Summary

- Use the HTML `` 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
- Use the HTML `<map>` element to define an image-map
- Use the HTML `<area>` element to define the clickable areas in the image-map
- Use the HTML ``'s element `usemap` attribute to point to an image-map
- Use the HTML `<picture>` element to show different images for different devices

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

HTML Image Tags

Tag	Description
<code></code>	Defines an image
<code><map></code>	Defines an image-map
<code><area></code>	Defines a clickable area inside an image-map
<code><picture></code>	Defines a container for multiple image resources

HTML Tables

HTML Table Example

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

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;  
}
```

Remember to define borders for both the table and the table cells.

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;  
}
```

Note: If the table has collapsed borders, `border-spacing` has no effect.

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>
```

HTML Table - Cells that Span Many Rows

To make a cell span more than one row, use the `rowspan` attribute:

Example

```
<table style="width:100%">
  <tr>
    <th>Name:</th>
    <td>Bill Gates</td>
  </tr>
  <tr>
    <th rowspan="2">Telephone:</th>
    <td>55577854</td>
  </tr>
  <tr>
    <td>55577855</td>
  </tr>
</table>
```

HTML Table - Adding a Caption

To add a caption to a table, use the `<caption>` tag:

Example

```
<table style="width:100%">
  <caption>Monthly savings</caption>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>February</td>
    <td>$50</td>
  </tr>
</table>
```

Note: The `<caption>` tag must be inserted immediately after the `<table>` tag.

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>
```

Now you can define a special style for this table:

```
table#t01 {
  width: 100%;
  background-color: #f1f1c1;
}
```

And add more styles:

```
table#t01 tr:nth-child(even) {
  background-color: #eee;
}
table#t01 tr:nth-child(odd) {
  background-color: #fff;
}
table#t01 th {
  color: white;
  background-color: black;
}
```

Chapter Summary

- Use the HTML `<table>` element to define a table
- Use the HTML `<tr>` element to define a table row
- Use the HTML `<td>` element to define a table data
- Use the HTML `<th>` element to define a table heading
- Use the HTML `<caption>` element to define a table caption
- Use the CSS `border` property to define a border
- Use the CSS `border-collapse` property to collapse cell borders
- Use the CSS `padding` property to add padding to cells
- Use the CSS `text-align` property to align cell text
- Use the CSS `border-spacing` property to set the spacing between cells
- Use the `colspan` attribute to make a cell span many columns
- Use the `rowspan` attribute to make a cell span many rows
- Use the `id` attribute to uniquely define one table

HTML Table Tags

Tag	Description
<code><table></code>	Defines a table
<code><th></code>	Defines a header cell in a table
<code><tr></code>	Defines a row in a table
<code><td></code>	Defines a cell in a table
<code><caption></code>	Defines a table caption
<code><colgroup></code>	Specifies a group of one or more columns in a table for formatting
<code><col></code>	Specifies column properties for each column within a <code><colgroup></code> element
<code><thead></code>	Groups the header content in a table
<code><tbody></code>	Groups the body content in a table
<code><tfoot></code>	Groups the footer content in a table

HTML Lists

HTML List Example

An Unordered List:

- Item
- Item
- Item
- Item

An Ordered List:

1. First item
2. Second item
3. Third item
4. Fourth item

Unordered HTML List

An unordered list starts with the `` tag. Each list item starts with the `` tag. The list items will be marked with bullets (small black circles) by default:

Example

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

Unordered HTML List - Choose List Item Marker

The CSS `list-style-type` property is used to define the style of the list item marker:

Value	Description
disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

Example - Disc

```
<ul style="list-style-type:disc">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Circle

```
<ul style="list-style-type:circle">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Square

```
<ul style="list-style-type:square">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - None

```
<ul style="list-style-type:none">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Ordered HTML List

An ordered list starts with the `` tag. Each list item starts with the `` tag.

The list items will be marked with numbers by default:

Example

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

Ordered HTML List - The Type Attribute

The `type` attribute of the `` tag, defines the type of the list item marker:

Type	Description
<code>type="1"</code>	The list items will be numbered with numbers (default)
<code>type="A"</code>	The list items will be numbered with uppercase letters
<code>type="a"</code>	The list items will be numbered with lowercase letters
<code>type="I"</code>	The list items will be numbered with uppercase roman numbers
<code>type="i"</code>	The list items will be numbered with lowercase roman numbers

Numbers:

```
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Letters:

```
<ol type="A">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Letters:

```
<ol type="a">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Roman Numbers:

```
<ol type="I">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Roman Numbers:

```
<ol type="i">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

HTML Description Lists

HTML also supports description lists.

A description list is a list of terms, with a description of each term.

The `<dl>` tag defines the description list, the `<dt>` tag defines the term (name), and the `<dd>` tag describes each term:

Example

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Nested HTML Lists

List can be nested (lists inside lists):

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
```

Note: List items can contain new list, and other HTML elements, like images and links, etc.

Control List Counting

By default, an ordered list will start counting from 1. If you want to start counting from a specified number, you can use the `start` attribute:

Example

```
<ol start="50">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Horizontal List with CSS

HTML lists can be styled in many different ways with CSS.

One popular way is to style a list horizontally, to create a navigation menu:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333333;
}

li {
    float: left;
}

li a {
    display: block;
    color: white;
    text-align: center;
    padding: 16px;
    text-decoration: none;
}

li a:hover {
    background-color: #111111;
}
</style>
</head>
<body>

<ul>
    <li><a href="#home">Home</a></li>
    <li><a href="#news">News</a></li>
    <li><a href="#contact">Contact</a></li>
    <li><a href="#about">About</a></li>
</ul>

</body>
</html>
```

Tip: You can learn much more about CSS in our [CSS Tutorial](#).

Chapter Summary

- Use the HTML `` element to define an unordered list
- Use the CSS `list-style-type` property to define the list item marker
- Use the HTML `` element to define an ordered list
- Use the HTML `type` attribute to define the numbering type
- Use the HTML `` element to define a list item
- Use the HTML `<dl>` element to define a description list
- Use the HTML `<dt>` element to define the description term
- Use the HTML `<dd>` element to describe the term in a description list
- Lists can be nested inside lists
- List items can contain other HTML elements
- Use the CSS property `float:left` or `display:inline` to display a list horizontally

HTML List Tags

Tag	Description
<code></code>	Defines an unordered list
<code></code>	Defines an ordered list
<code></code>	Defines a list item
<code><dl></code>	Defines a description list
<code><dt></code>	Defines a term in a description list
<code><dd></code>	Describes the term in a description list

HTML Block and Inline Elements

Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline.

Block-level Elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

The `<div>` element is a block-level element.

Example

```
<div>Hello</div>
<div>World</div>
```

Block level elements in HTML:

<address>
<article>
<aside>
<blockquote>
<canvas>
<dd>
<div>
<dl>
<dt>
<fieldset>
<figcaption>
<figure>
<footer>
<form>
<h1>-<h6>
<header>
<hr>

<main>
<nav>
<noscript>

<output>
<p>
<pre>
<section>
<table>
<tfoot>

<video>

Inline Elements

An inline element does not start on a new line and only takes up as much width as necessary.
This is an inline element inside a paragraph.

Example

```
<span>Hello</span>
<span>World</span>
```

Inline elements in HTML:

<a>
<abbr>
<acronym>

<bdo>
<big>

<button>
<cite>
<code>
<dfn>

<i>

<input>
<kbd>
<label>
<map>
<object>
<q>
<samp>
<script>
<select>
<small>

<sub>
<sup>
<textarea>
<time>
<tt>
<var>

The <div> Element

The `<div>` element is often used as a container for other HTML elements.

The `<div>` element has no required attributes, but `style`, `class` and `id` are common.

When used together with CSS, the `<div>` element can be used to style blocks of content:

Example

```
<div style="background-color:black;color:white;padding:20px;">
  <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>
</div>
```

The Element

The `` element is often used as a container for some text.

The `` element has no required attributes, but `style`, `class` and `id` are common.

When used together with CSS, the `` element can be used to style parts of the text:

Example

```
<h1>My <span style="color:red">Important</span> Heading</h1>
```

HTML Grouping Tags

Tag	Description
<code><div></code>	Defines a section in a document (block-level)
<code></code>	Defines a section in a document (inline)

HTML The class Attribute

Using The class Attribute

The `class` attribute specifies one or more class names for an HTML element.

In CSS, to select elements with a specific class, write a period (.) character, followed by the name of the class:

Tip: The class attribute can be used on **any** HTML element.

Note: The class name is case sensitive!

Tip: You can learn much more about CSS in our [CSS Tutorial](#).

Using The class Attribute in JavaScript

JavaScript can access elements with a specified class name by using the `getElementsByClassName()` method:

Example

When a user clicks on a button, hide all elements with the class name "city":

```
<script>
function myFunction() {
  var x = document.getElementsByClassName("city");
  for (var i = 0; i < x.length; i++) {
    x[i].style.display = "none";
  }
}
</script>
```

Tip: Study JavaScript in the chapter [HTML JavaScript](#), or in our [JavaScript Tutorial](#).

Multiple Classes

HTML elements can have more than one class name, each class name must be separated by a space.

Example

Style elements with the class name "city", also style elements with the class name "main":

```
<h2 class="city main">London</h2>
<h2 class="city">Paris</h2>
<h2 class="city">Tokyo</h2>
```

In the example above, the first `<h2>` element belongs to both the "city" class and the "main" class.

Same Class, Different Tag

Different tags, like `<h2>` and `<p>`, can have the same class name and thereby share the same style:

Example

```
<h2 class="city">Paris</h2>
<p class="city">Paris is the capital of France</p>
```

HTML The id Attribute

Using The id Attribute

The `id` attribute specifies a unique id for an HTML element (the value must be unique within the HTML document).

The id value can be used by CSS and JavaScript to perform certain tasks for a unique element with the specified id value.

In CSS, to select an element with a specific id, write a hash (#) character, followed by the id of the element:

Example

Use CSS to style an element with the id "myHeader":

```
<style>
#myHeader {
    background-color: lightblue;
    color: black;
    padding: 40px;
    text-align: center;
}
</style>

<h1 id="myHeader">My Header</h1>
```

Result:

My Header

Tip: The id attribute can be used on **any** HTML element.

Note: The id value is case-sensitive.

Note: The id value must contain at least **one** character, and must **not** contain whitespace (spaces, tabs, etc.).

Difference Between Class and ID

An HTML element can only have one unique id that belongs to that single element, while a class name can be used by multiple elements:

Example

```
<style>
/* Style the element with the id "myHeader" */
#myHeader {
    background-color: lightblue;
    color: black;
    padding: 40px;
    text-align: center;
}

/* Style all elements with the class name "city" */
.city {
    background-color: tomato;
    color: white;
    padding: 10px;
}
</style>

<!-- A unique element -->
<h1 id="myHeader">My Cities</h1>

<!-- Multiple similar elements -->
<h2 class="city">London</h2>
<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>
<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>
<p>Tokyo is the capital of Japan.</p>
```

Tip: You can learn much more about CSS in our [CSS Tutorial](#).

Using The id Attribute in JavaScript

JavaScript can access an element with a specified id by using the `getElementById()` method:

Example

Use the id attribute to manipulate text with JavaScript:

```
<script>
function displayResult() {
    document.getElementById("myHeader").innerHTML = "Have a nice day!";
}
</script>
```

Tip: Study JavaScript in the chapter [HTML JavaScript](#), or in our [JavaScript Tutorial](#).

Bookmarks with ID and Links

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

Bookmarks can be useful if your webpage is very long.

To make a bookmark, you must first create the bookmark, and then add a link to it.

When the link is clicked, the page will scroll 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:

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

Or, add a link to the bookmark ("Jump to Chapter 4"), from another page:

Example

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

HTML JavaScript

JavaScript makes HTML pages more dynamic and interactive.

Example

My First JavaScript

Click me to display Date and Time

The HTML <script> Tag

The `<script>` tag is used to define a client-side script (JavaScript).

The `<script>` element either contains scripting statements, or it points to an external script file through the `src` attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

To select an HTML element, JavaScript very often uses the `document.getElementById()` method.

This JavaScript example writes "Hello JavaScript!" into an HTML element with id="demo":

Example

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
```

Tip: You can learn much more about JavaScript in our [JavaScript Tutorial](#).

A Taste of JavaScript

Here are some examples of what JavaScript can do:

JavaScript can change HTML content

```
document.getElementById("demo").innerHTML = "Hello JavaScript!";
```

JavaScript can change HTML styles

```
document.getElementById("demo").style.fontSize = "25px";
document.getElementById("demo").style.color = "red";
document.getElementById("demo").style.backgroundColor = "yellow";
```

JavaScript can change HTML attributes

```
document.getElementById("image").src = "picture.gif";
```

The HTML <noscript> Tag

The `<noscript>` tag is used to provide an alternate content for users that have disabled scripts in their browser or have a browser that doesn't support client-side scripts:

Example

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>

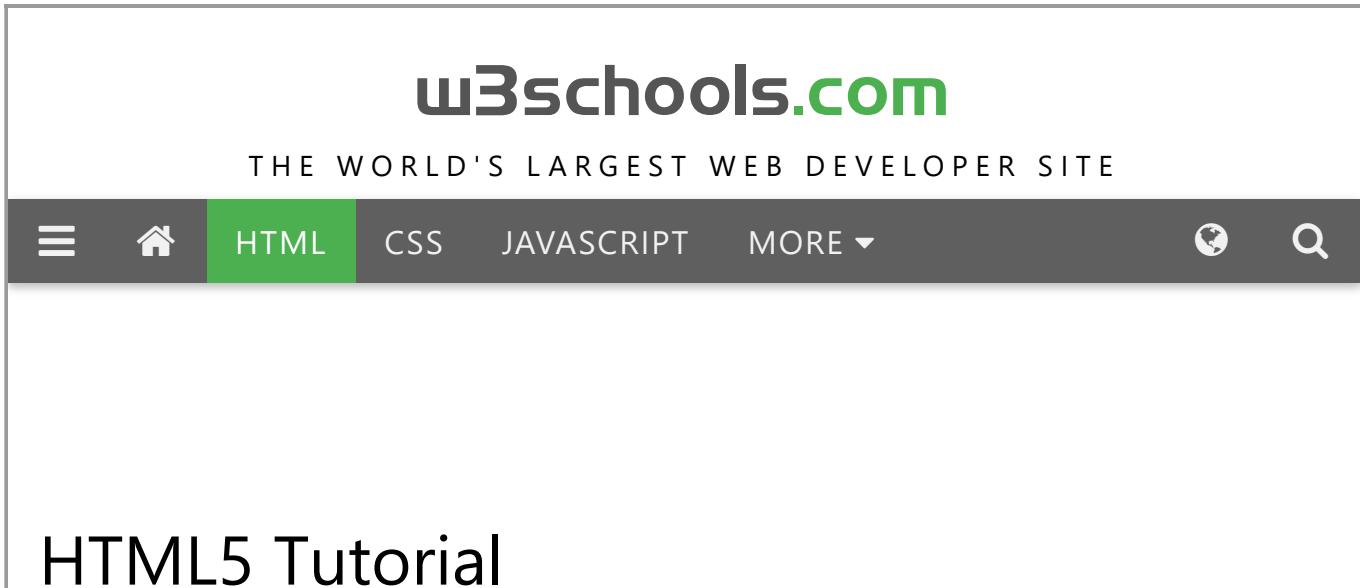
<noscript>Sorry, your browser does not support JavaScript!</noscript>
```

HTML Script Tags

Tag	Description
<code><script></code>	Defines a client-side script
<code><noscript></code>	Defines an alternate content for users that do not support client-side scripts

HTML Iframes

An iframe is used to display a web page within a web page.

A screenshot of the w3schools.com website. At the top, the w3schools.com logo is displayed in green and grey. Below it, the text "THE WORLD'S LARGEST WEB DEVELOPER SITE" is centered. A navigation bar follows, featuring icons for a menu, home, and search, along with links for "HTML", "CSS", "JAVASCRIPT", and "MORE". The "HTML" link is highlighted with a green background. The main content area below the navigation bar displays the title "HTML5 Tutorial".

Iframe Syntax

An HTML iframe is defined with the `<iframe>` tag:

```
<iframe src="URL"></iframe>
```

The `src` attribute specifies the URL (web address) of the inline frame page.

Iframe - Set Height and Width

Use the `height` and `width` attributes to specify the size of the iframe.

The attribute values are specified in pixels by default, but they can also be in percent (like "80%").

Example

```
<iframe src="demo_iframe.htm" height="200" width="300"></iframe>
```

Or you can use CSS to set the height and width of the iframe:

Example

```
<iframe src="demo_iframe.htm" style="height:200px;width:300px;"></iframe>
```

Iframe - Remove the Border

By default, an iframe has a border around it.

To remove the border, add the `style` attribute and use the CSS `border` property:

Example

```
<iframe src="demo_iframe.htm" style="border:none;"></iframe>
```

With CSS, you can also change the size, style and color of the iframe's border:

Example

```
<iframe src="demo_iframe.htm" style="border:2px solid red;"></iframe>
```

Iframe - Target for a Link

An iframe can be used as the target frame for a link.

The `target` attribute of the link must refer to the `name` attribute of the iframe:

Example

```
<iframe src="demo_iframe.htm" name="iframe_a"></iframe>

<p><a href="https://www.w3schools.com" target="iframe_a">W3Schools.com</a>
</p>
```

Tag	Description
<code><iframe></code>	Defines an inline frame

HTML File Paths

Path	Description
	picture.jpg is located in the same folder as the current page
	picture.jpg is located in the images folder in the current folder
	picture.jpg is located in the images folder at the root of the current web
	picture.jpg is located in the folder one level up from the current folder

HTML File Paths

A file path describes the location of a file in a web site's folder structure.

File paths are used when linking to external files like:

- Web pages
- Images
- Style sheets
- JavaScripts

Absolute File Paths

An absolute file path is the full URL to an internet file:

Example

```

```

The tag and the src and alt attributes are explained in the chapter about [HTML Images](#).

Relative File Paths

A relative file path points to a file relative to the current page.

In this example, the file path points to a file in the images folder located at the root of the current web:

Example

```

```

In this example, the file path points to a file in the images folder located in the current folder:

Example

```

```

In this example, the file path points to a file in the images folder located in the folder one level above the current folder:

Example

```

```

Best Practice

It is best practice to use relative file paths (if possible).

When using relative file paths, your web pages will not be bound to your current base URL. All links will work on your own computer (localhost) as well as on your current public domain and your future public domains.

HTML Head

The HTML <head> Element

The `<head>` element is a container for metadata (data about data) and is placed between the `<html>` tag and the `<body>` tag.

HTML metadata is data about the HTML document. Metadata is not displayed.

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

The following tags describe metadata: `<title>`, `<style>`, `<meta>`, `<link>`, `<script>`, and `<base>`.

The HTML <title> Element

The `<title>` element defines the title of the document, and is required in all HTML/XHTML documents.

The `<title>` element:

- defines a title in the browser tab
- provides a title for the page when it is added to favorites
- displays a title for the page in search engine results

A simple HTML document:

Example

```
<!DOCTYPE html>
<html>

<head>
  <title>Page Title</title>
</head>

<body>
The content of the document.....
</body>

</html>
```

The HTML <style> Element

The `<style>` element is used to define style information for a single HTML page:

Example

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

The HTML <link> Element

The `<link>` element is used to link to external style sheets:

Example

```
<link rel="stylesheet" href="mystyle.css">
```

Tip: To learn all about CSS, visit our [CSS Tutorial](#).

The HTML <meta> Element

The `<meta>` element is used to specify which character set is used, page description, keywords, author, and other metadata.

Metadata is used by browsers (how to display content), by search engines (keywords), and other web services.

Define the character set used:

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

Define a description of your web page:

```
<meta name="description" content="Free Web tutorials">
```

Define keywords for search engines:

```
<meta name="keywords" content="HTML, CSS, XML, JavaScript">
```

Define the author of a page:

```
<meta name="author" content="John Doe">
```

Refresh document every 30 seconds:

```
<meta http-equiv="refresh" content="30">
```

Example of `<meta>` tags:

Example

```
<meta charset="UTF-8">
<meta name="description" content="Free Web tutorials">
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<meta name="author" content="John Doe">
```

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the `<meta>` tag.

The viewport is the user's visible area of a web page. It varies with the device, and will be smaller on a mobile phone than on a computer screen.

You should include the following `<meta>` viewport element in all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

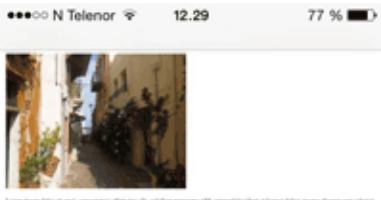
A `<meta>` viewport element gives the browser instructions on how to control the page's dimensions and scaling.

The `width=device-width` part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The `initial-scale=1.0` part sets the initial zoom level when the page is first loaded by the browser.

Here is an example of a web page *without* the viewport meta tag, and the same web page *with* the viewport `<meta>` tag:

Tip: If you are browsing this page with a phone or a tablet, you can click on the two links below to see the difference.



Without the

viewport meta tag



With the viewport

meta tag

Lorum ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option conone nihil imperdiet domino...

The HTML <script> Element

The `<script>` element is used to define client-side JavaScripts.

This JavaScript writes "Hello JavaScript!" into an HTML element with id="demo":

Example

```
<script>
function myFunction {
    document.getElementById("demo").innerHTML = "Hello JavaScript!";
}
</script>
```

Tip: To learn all about JavaScript, visit our [JavaScript Tutorial](#).

The HTML <base> Element

The `<base>` element specifies the base URL and base target for all relative URLs in a page:

Example

```
<base href="https://www.w3schools.com/images/" target="_blank">
```

Omitting <html>, <head> and <body>?

According to the HTML5 standard; the `<html>`, the `<body>`, and the `<head>` tag can be omitted.

The following code will validate as HTML5:

Example

```
<!DOCTYPE html>
<title>Page Title</title>

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

Note:

W3Schools does not recommend omitting the `<html>` and `<body>` tags. Omitting these tags can crash DOM or XML software and produce errors in older browsers (IE9). However, omitting the `<head>` tag has been a common practice for quite some time now.

HTML head Elements

Tag	Description
<code><head></code>	Defines information about the document
<code><title></code>	Defines the title of a document
<code><base></code>	Defines a default address or a default target for all links on a page
<code><link></code>	Defines the relationship between a document and an external resource
<code><meta></code>	Defines metadata about an HTML document
<code><script></code>	Defines a client-side script
<code><style></code>	Defines style information for a document

HTML Layouts

HTML Layout Example

Cities

- [London](#)
- [Paris](#)
- [Tokyo](#)

London

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.

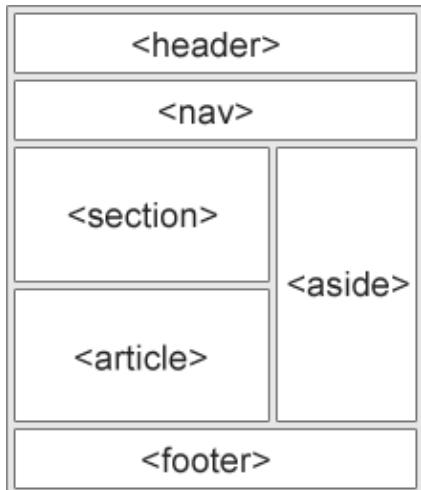
Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.

Footer

HTML Layout Elements

Websites often display content in multiple columns (like a magazine or newspaper).

HTML5 offers new semantic elements that define the different parts of a web page:



- `<header>` - Defines a header for a document or a section
- `<nav>` - Defines a container for navigation links
- `<section>` - Defines a section in a document
- `<article>` - Defines an independent self-contained article
- `<aside>` - Defines content aside from the content (like a sidebar)
- `<footer>` - Defines a footer for a document or a section
- `<details>` - Defines additional details
- `<summary>` - Defines a heading for the `<details>` element

HTML Layout Techniques

There are four different ways to create multicolumn layouts. Each way has its pros and cons:

- HTML tables (not recommended)
- CSS float property
- CSS flexbox
- CSS framework

Which One to Choose?

HTML Tables

The `<table>` element was not designed to be a layout tool! The purpose of the `<table>` element is to display tabular data. So, do not use tables for your page layout! They will bring a mess into your code. And imagine how hard it will be to redesign your site after a couple of months.

Tip: Do NOT use tables for your page layout!

CSS Frameworks

If you want to create your layout fast, you can use a framework, like [W3.CSS](#) or [Bootstrap](#).

CSS Floats

It is common to do entire web layouts using the CSS float property. Float is easy to learn - you just need to remember how the float and clear properties work. **Disadvantages:** Floating elements are tied to the document flow, which may harm the flexibility. Learn more about float in our [CSS Float and Clear](#) chapter.

Float Example

Cities

- [London](#)
- [Paris](#)
- [Tokyo](#)

London

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.

Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.

Footer

CSS Flexbox

Flexbox is a new layout mode in CSS3.

Use of flexbox ensures that elements behave predictably when the page layout must accommodate different screen sizes and different display devices. **Disadvantages:** Does not work in IE10 and earlier.

Learn more about flexbox in our [CSS Flexbox](#) chapter.

Flexbox Example

Cities

- [London](#)
- [Paris](#)
- [Tokyo](#)

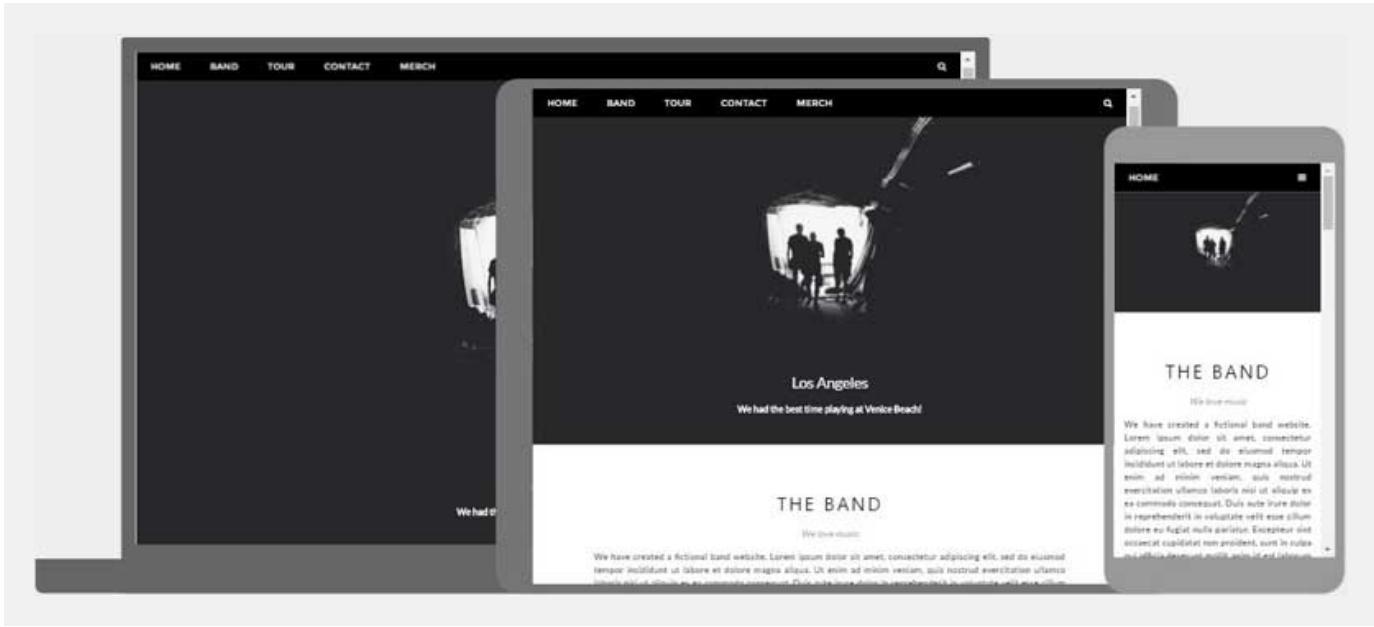
London

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.

Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.

Footer

HTML Responsive Web Design



What is Responsive Web Design?

Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones):

[Try it Yourself >](#)

Note: A web page should look good on **any device!**

Setting The Viewport

When making responsive web pages, add the following `<meta>` element in all your web pages:

Example

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.

Here is an example of a web page *without* the viewport meta tag, and the same web page *with* the viewport meta tag:

Without the viewport meta tag:



Etiam velit euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet domino.

With the viewport meta tag:



Etiam velit euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet domino.

Tip: If you are browsing this page on a phone or a tablet, you can click on the two links above to see the difference.

Responsive Images

Responsive images are images that scale nicely to fit any browser size.

Using the width Property

If the CSS `width` property is set to 100%, the image will be responsive and scale up and down:



Example

```

```

Notice that in the example above, the image can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the `max-width` property instead.

Using the max-width Property

If the `max-width` property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:



Example

```

```

Show Different Images Depending on Browser Width

The HTML `<picture>` element allows you to define different images for different browser window sizes.

Resize the browser window to see how the image below change depending on the width:



Example

```
<picture>
  <source srcset="img_smallflower.jpg" media="(max-width: 600px)">
  <source srcset="img_flowers.jpg" media="(max-width: 1500px)">
  <source srcset="flowers.jpg">
  
</picture>
```

Responsive Text Size

The text size can be set with a "vw" unit, which means the "viewport width". That way the text size will follow the size of the browser window:

Hello World

Resize the browser window to see how the text size scales.

Example

```
<h1 style="font-size:10vw">Hello World</h1>
```

Viewport is the browser window size. 1vw = 1% of viewport width. If the viewport is 50cm wide, 1vw is 0.5cm.

Media Queries

In addition to resize text and images, it is also common to use media queries in responsive web pages.

With media queries you can define completely different styles for different browser sizes.

Example: resize the browser window to see that the three div elements below will display horizontally on large screens and stacked vertically on small screens:



Example

```
<style>
.left, .right {
  float: left;
  width: 20%; /* The width is 20%, by default */
}

.main {
  float: left;
  width: 60%; /* The width is 60%, by default */
}

/* Use a media query to add a breakpoint at 800px: */
@media screen and (max-width: 800px) {
  .left, .main, .right {
    width: 100%; /* The width is 100%, when the viewport is 800px or
smaller */
  }
}
</style>
```

Tip: To learn more about Media Queries and Responsive Web Design, read our [RWD Tutorial](#).

Responsive Web Page - Full Example

A responsive web page should look good on large desktop screens and small mobile phones.

Try it Yourself »

Responsive Web Design - Frameworks

There are many existing CSS Frameworks that offer Responsive Design.
They are free, and easy to use.

Using W3.CSS

A great way to create a responsive design, is to use a responsive style sheet, like [W3.CSS](#)
W3.CSS makes it easy to develop sites that look nice at any size; desktop, laptop, tablet, or phone:

W3.CSS Demo

Resize the page to see the responsiveness!

London

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.

Paris

Paris is the capital of France. The Paris area is one of the largest population centers in Europe, with more than 12 million inhabitants.

Tokyo

Tokyo is the capital of Japan. It is the center of the Greater Tokyo Area, and the most populous metropolitan area in the world.

Example

```
<!DOCTYPE html>
<html>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
<body>

<div class="w3-container w3-green">
  <h1>W3Schools Demo</h1>
  <p>Resize this responsive page!</p>
</div>

<div class="w3-row-padding">
  <div class="w3-third">
    <h2>London</h2>
    <p>London is the capital city of England.</p>
    <p>It is the most populous city in the United Kingdom,
    with a metropolitan area of over 13 million inhabitants.</p>
  </div>

  <div class="w3-third">
    <h2>Paris</h2>
    <p>Paris is the capital of France.</p>
    <p>The Paris area is one of the largest population centers in Europe,
    with more than 12 million inhabitants.</p>
  </div>

  <div class="w3-third">
    <h2>Tokyo</h2>
    <p>Tokyo is the capital of Japan.</p>
    <p>It is the center of the Greater Tokyo Area,
    and the most populous metropolitan area in the world.</p>
  </div>
</div>

</body>
</html>
```

To learn more about W3.CSS, read our [W3.CSS Tutorial](#).

Bootstrap

Another popular framework is Bootstrap, it uses HTML, CSS and jQuery to make responsive web pages.

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Bootstrap Example</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script
      src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
</script>
<script
      src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js">
</script>
</head>
<body>

<div class="container">
  <div class="jumbotron">
    <h1>My First Bootstrap Page</h1>
  </div>
  <div class="row">
    <div class="col-sm-4">
      ...
    </div>
    <div class="col-sm-4">
      ...
    </div>
    <div class="col-sm-4">
      ...
    </div>
  </div>
</div>

</body>
</html>
```

To learn more about Bootstrap, go to our [Bootstrap Tutorial](#).

HTML Computer Code Elements

Computer Code

```
<code>
x = 5;<br>
y = 6;<br>
z = x + y;
</code>
```

HTML <kbd> For Keyboard Input

The HTML `<kbd>` element represents user input, like keyboard input or voice commands. Text surrounded by `<kbd>` tags is typically displayed in the browser's default monospace font:

Example

```
<p>Save the document by pressing <kbd>Ctrl + S</kbs></p>
```

Result:

Save the document by pressing Ctrl + S

HTML <samp> For Program Output

The HTML `<samp>` element represents output from a program or computing system. Text surrounded by `<samp>` tags is typically displayed in the browser's default monospace font:

Example

```
<p>If you input wrong value, the program will return <samp>Error!</smp>
</p>
```

Result:

If you input wrong value, the program will return Error!

HTML <code> For Computer Code

The HTML `<code>` element defines a fragment of computer code.

Text surrounded by `<code>` tags is typically displayed in the browser's default monospace font:

Example

```
<code>
x = 5;
y = 6;
z = x + y;
</code>
```

Result:

```
x = 5; y = 6; z = x + y;
```

Notice that the `<code>` element does not preserve extra whitespace and line-breaks.

To fix this, you can put the `<code>` element inside a `<pre>` element:

Example

```
<pre>
<code>
x = 5;
y = 6;
z = x + y;
</code>
</pre>
```

Result:

```
x = 5;
y = 6;
z = x + y;
```

HTML `<var>` For Variables

The HTML `<var>` element defines a variable.

The variable could be a variable in a mathematical expression or a variable in programming context:

Example

Einstein wrote: `<var>E</var> = <var>mc</var>².`

Result:

Einstein wrote: $E = mc^2$.

HTML Computer Code Elements

Tag	Description
<code><code></code>	Defines programming code
<code><kbd></code>	Defines keyboard input
<code><samp></code>	Defines computer output
<code><var></code>	Defines a variable
<code><pre></code>	Defines preformatted text

HTML Entities

Reserved characters in HTML must be replaced with character entities.

Characters that are not present on your keyboard can also be replaced by entities.

HTML Entities

Some characters are reserved in HTML.

If you use the less than (<) or greater than (>) signs in your text, the browser might mix them with tags.

Character entities are used to display reserved characters in HTML.

A character entity looks like this:

```
&entity_name;  
OR  
&#entity_number;
```

To display a less than sign (<) we must write: **<** or **<**

Advantage of using an entity name: An entity name is easy to remember.

Disadvantage of using an entity name: Browsers may not support all entity names, but the support for numbers is good.

Non-breaking Space

A common character entity used in HTML is the non-breaking space: **&nbsp**

A non-breaking space is a space that will not break into a new line.

Two words separated by a non-breaking space will stick together (not break into a new line).

This is handy when breaking the words might be disruptive.

Examples:

- § 10
- 10 km/h
- 10 PM

Another common use of the non-breaking space is to prevent browsers from truncating spaces in HTML pages.

If you write 10 spaces in your text, the browser will remove 9 of them. To add real spaces to your text, you can use the **&nbsp** character entity.

The non-breaking hyphen (‑) lets you use a hyphen character (-) that won't break.

Some Other Useful HTML Character Entities

Result	Description	Entity Name	Entity Number
	non-breaking space	 	
<	less than	<	<
>	greater than	>	>
&	ampersand	&	&
"	double quotation mark	"	"
'	single quotation mark (apostrophe)	'	'
¢	cent	¢	¢
£	pound	£	£
¥	yen	¥	¥
€	euro	€	€
©	copyright	©	©
®	registered trademark	®	®

Note: Entity names are case sensitive.

Combining Diacritical Marks

A diacritical mark is a "glyph" added to a letter.

Some diacritical marks, like grave (`) and acute (') are called accents.

Diacritical marks can appear both above and below a letter, inside a letter, and between two letters.

Diacritical marks can be used in combination with alphanumeric characters to produce a character that is not present in the character set (encoding) used in the page.

Here are some examples:

Mark	Character	Construct	Result
'	a	à	à
'	a	á	á
^	a	â	â
~	a	ã	ã
'	o	Ò	ò
'	o	Ó	ó
^	o	Ô	ô
~	o	Õ	õ

You will see more HTML symbols in the next chapter of this tutorial.

HTML Symbols

HTML Symbol Entities

HTML entities were described in the previous chapter.

Many mathematical, technical, and currency symbols, are not present on a normal keyboard.

To add such symbols to an HTML page, you can use an HTML entity name.

If no entity name exists, you can use an entity number, a decimal, or hexadecimal reference.

Example

```
<p>I will display &euro;</p>
<p>I will display &#8364;</p>
<p>I will display &#x20AC;</p>
```

Will display as:

I will display €
I will display €
I will display €

Some Mathematical Symbols Supported by HTML

Char	Number	Entity	Description
∀	∀	∀	FOR ALL
∂	∂	∂	PARTIAL DIFFERENTIAL
∃	∃	∃	THERE EXISTS
∅	∅	∅	EMPTY SETS
∇	∇	∇	NABLA
∈	∈	∈	ELEMENT OF
∉	∉	∉	NOT AN ELEMENT OF
∋	∋	∋	CONTAINS AS MEMBER
∏	∏	∏	N-ARY PRODUCT
Σ	∑	∑	N-ARY SUMMATION

[Full Math Reference](#)

Some Greek Letters Supported by HTML

Char	Number	Entity	Description
Α	Α	Α	GREEK CAPITAL LETTER ALPHA
Β	Β	Β	GREEK CAPITAL LETTER BETA
Γ	Γ	Γ	GREEK CAPITAL LETTER GAMMA
Δ	Δ	Δ	GREEK CAPITAL LETTER DELTA
Ε	Ε	Ε	GREEK CAPITAL LETTER EPSILON
Ζ	Ζ	Ζ	GREEK CAPITAL LETTER ZETA

[Full Greek Reference](#)

Some Other Entities Supported by HTML

Char	Number	Entity	Description
©	©	©	COPYRIGHT SIGN
®	®	®	REGISTERED SIGN
€	€	€	EURO SIGN
™	™	™	TRADEMARK
←	←	←	LEFTWARDS ARROW
↑	↑	↑	UPWARDS ARROW
→	→	→	RIGHTWARDS ARROW
↓	↓	↓	DOWNWARDS ARROW
♠	♠	♠	BLACK SPADE SUIT
♣	♣	♣	BLACK CLUB SUIT
♥	♥	♥	BLACK HEART SUIT
♦	♦	♦	BLACK DIAMOND SUIT

[Full Currency Reference](#)

[Full Arrows Reference](#)

[Full Symbols Reference](#)

HTML Encoding (Character Sets)

To display an HTML page correctly, a web browser must know which character set (character encoding) to use.

What is Character Encoding?

ASCII was the first **character encoding standard** (also called character set). ASCII defined 128 different alphanumeric characters that could be used on the internet: numbers (0-9), English letters (A-Z), and some special characters like ! \$ + - () @ < > .

ANSI (Windows-1252) was the original Windows character set, with support for 256 different character codes.

ISO-8859-1 was the default character set for HTML 4. This character set also supported 256 different character codes.

Because ANSI and ISO-8859-1 were so limited, HTML 4 also supported UTF-8.

UTF-8 (Unicode) covers almost all of the characters and symbols in the world.

The default character encoding for HTML5 is UTF-8.

The HTML charset Attribute

To display an HTML page correctly, a web browser must know the character set used in the page.

This is specified in the `<meta>` tag:

For HTML4:

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

For HTML5:

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

If a browser detects ISO-8859-1 in a web page, it defaults to ANSI, because ANSI is identical to ISO-8859-1 except that ANSI has 32 extra characters.

Differences Between Character Sets

The following table displays the differences between the character sets described above:

Numb	ASCII	ANSI	8859	UTF-	Description
8					

32					space
33	!	!	!	!	exclamation mark
34	"	"	"	"	quotation mark
35	#	#	#	#	number sign
36	\$	\$	\$	\$	dollar sign
37	%	%	%	%	percent sign
38	&	&	&	&	ampersand
39	'	'	'	'	apostrophe
40	((((left parenthesis
41))))	right parenthesis
42	*	*	*	*	asterisk
43	+	+	+	+	plus sign
44	,	,	,	,	comma
45	-	-	-	-	hyphen-minus
46	full stop
47	/	/	/	/	solidus
48	0	0	0	0	digit zero
49	1	1	1	1	digit one
50	2	2	2	2	digit two
51	3	3	3	3	digit three
52	4	4	4	4	digit four
53	5	5	5	5	digit five
54	6	6	6	6	digit six
55	7	7	7	7	digit seven
56	8	8	8	8	digit eight
57	9	9	9	9	digit nine
58	:	:	:	:	colon
59	;	;	;	;	semicolon

60	<	<	<	<	less-than sign
61	=	=	=	=	equals sign
62	>	>	>	>	greater-than sign
63	?	?	?	?	question mark
64	@	@	@	@	commercial at
65	A	A	A	A	Latin capital letter A
66	B	B	B	B	Latin capital letter B
67	C	C	C	C	Latin capital letter C
68	D	D	D	D	Latin capital letter D
69	E	E	E	E	Latin capital letter E
70	F	F	F	F	Latin capital letter F
71	G	G	G	G	Latin capital letter G
72	H	H	H	H	Latin capital letter H
73	I	I	I	I	Latin capital letter I
74	J	J	J	J	Latin capital letter J
75	K	K	K	K	Latin capital letter K
76	L	L	L	L	Latin capital letter L
77	M	M	M	M	Latin capital letter M
78	N	N	N	N	Latin capital letter N
79	O	O	O	O	Latin capital letter O
80	P	P	P	P	Latin capital letter P
81	Q	Q	Q	Q	Latin capital letter Q
82	R	R	R	R	Latin capital letter R
83	S	S	S	S	Latin capital letter S
84	T	T	T	T	Latin capital letter T
85	U	U	U	U	Latin capital letter U
86	V	V	V	V	Latin capital letter V
87	W	W	W	W	Latin capital letter W

88	X	X	X	X	Latin capital letter X
89	Y	Y	Y	Y	Latin capital letter Y
90	Z	Z	Z	Z	Latin capital letter Z
91	[[[[left square bracket
92	\	\	\	\	reverse solidus
93]]]]	right square bracket
94	^	^	^	^	circumflex accent
95	-	-	-	-	low line
96	`	`	`	`	grave accent
97	a	a	a	a	Latin small letter a
98	b	b	b	b	Latin small letter b
99	c	c	c	c	Latin small letter c
100	d	d	d	d	Latin small letter d
101	e	e	e	e	Latin small letter e
102	f	f	f	f	Latin small letter f
103	g	g	g	g	Latin small letter g
104	h	h	h	h	Latin small letter h
105	i	i	i	i	Latin small letter i
106	j	j	j	j	Latin small letter j
107	k	k	k	k	Latin small letter k
108	l	l	l	l	Latin small letter l
109	m	m	m	m	Latin small letter m
110	n	n	n	n	Latin small letter n
111	o	o	o	o	Latin small letter o
112	p	p	p	p	Latin small letter p
113	q	q	q	q	Latin small letter q
114	r	r	r	r	Latin small letter r
115	s	s	s	s	Latin small letter s

116	t	t	t	t	Latin small letter t
117	u	u	u	u	Latin small letter u
118	v	v	v	v	Latin small letter v
119	w	w	w	w	Latin small letter w
120	x	x	x	x	Latin small letter x
121	y	y	y	y	Latin small letter y
122	z	z	z	z	Latin small letter z
123	{	{	{	{	left curly bracket
124					vertical line
125	}	}	}	}	right curly bracket
126	~	~	~	~	tilde
127	DEL				
128	€				euro sign
129	□	□	□		NOT USED
130	,				single low-9 quotation mark
131	f				Latin small letter f with hook
132	„				double low-9 quotation mark
133	…				horizontal ellipsis
134	†				dagger
135	‡				double dagger
136	^				modifier letter circumflex accent
137	%o				per mille sign
138	Š				Latin capital letter S with caron
139	<				single left-pointing angle quotation mark
140	Œ				Latin capital ligature OE
141	□	□	□		NOT USED
142	Ž				Latin capital letter Z with caron
143	□	□	□		NOT USED

144	□	□	□	NOT USED
145	‘			left single quotation mark
146	’			right single quotation mark
147	“			left double quotation mark
148	”			right double quotation mark
149	•			bullet
150	–			en dash
151	—			em dash
152	~			small tilde
153	™			trade mark sign
154	š			Latin small letter s with caron
155	>			single right-pointing angle quotation mark
156	œ			Latin small ligature oe
157	□	□	□	NOT USED
158	ž			Latin small letter z with caron
159	Ÿ			Latin capital letter Y with diaeresis
160				no-break space
161	¡	¡	¡	inverted exclamation mark
162	¢	¢	¢	cent sign
163	£	£	£	pound sign
164	¤	¤	¤	currency sign
165	¥	¥	¥	yen sign
166				broken bar
167	§	§	§	section sign
168	diaeresis
169	©	©	©	copyright sign
170	ª	ª	ª	feminine ordinal indicator
171	«	«	«	left-pointing double angle quotation mark

172	¬	¬	¬	not sign
173				soft hyphen
174	®	®	®	registered sign
175	—	—	—	macron
176	°	°	°	degree sign
177	±	±	±	plus-minus sign
178	²	²	²	superscript two
179	³	³	³	superscript three
180	'	'	'	acute accent
181	µ	µ	µ	micro sign
182	¶	¶	¶	pilcrow sign
183	·	·	·	middle dot
184	¸	¸	¸	cedilla
185	¹	¹	¹	superscript one
186	º	º	º	masculine ordinal indicator
187	»	»	»	right-pointing double angle quotation mark
188	¼	¼	¼	vulgar fraction one quarter
189	½	½	½	vulgar fraction one half
190	¾	¾	¾	vulgar fraction three quarters
191	¿	¿	¿	inverted question mark
192	À	À	À	Latin capital letter A with grave
193	Á	Á	Á	Latin capital letter A with acute
194	Â	Â	Â	Latin capital letter A with circumflex
195	Ã	Ã	Ã	Latin capital letter A with tilde
196	Ä	Ä	Ä	Latin capital letter A with diaeresis
197	Å	Å	Å	Latin capital letter A with ring above
198	Æ	Æ	Æ	Latin capital letter AE
199	Ҫ	Ҫ	Ҫ	Latin capital letter C with cedilla

200	È	È	È	Latin capital letter E with grave
201	É	É	É	Latin capital letter E with acute
202	Ê	Ê	Ê	Latin capital letter E with circumflex
203	Ë	Ë	Ë	Latin capital letter E with diaeresis
204	Ì	Ì	Ì	Latin capital letter I with grave
205	Í	Í	Í	Latin capital letter I with acute
206	Î	Î	Î	Latin capital letter I with circumflex
207	Ï	Ï	Ï	Latin capital letter I with diaeresis
208	Ð	Ð	Ð	Latin capital letter Eth
209	Ñ	Ñ	Ñ	Latin capital letter N with tilde
210	Ò	Ò	Ò	Latin capital letter O with grave
211	Ó	Ó	Ó	Latin capital letter O with acute
212	Ô	Ô	Ô	Latin capital letter O with circumflex
213	Õ	Õ	Õ	Latin capital letter O with tilde
214	Ö	Ö	Ö	Latin capital letter O with diaeresis
215	×	×	×	multiplication sign
216	Ø	Ø	Ø	Latin capital letter O with stroke
217	Ù	Ù	Ù	Latin capital letter U with grave
218	Ú	Ú	Ú	Latin capital letter U with acute
219	Û	Û	Û	Latin capital letter U with circumflex
220	Ü	Ü	Ü	Latin capital letter U with diaeresis
221	Ý	Ý	Ý	Latin capital letter Y with acute
222	Þ	Þ	Þ	Latin capital letter Thorn
223	ß	ß	ß	Latin small letter sharp s
224	à	à	à	Latin small letter a with grave
225	á	á	á	Latin small letter a with acute
226	â	â	â	Latin small letter a with circumflex
227	ã	ã	ã	Latin small letter a with tilde

228	ä	ä	ä	Latin small letter a with diaeresis
229	å	å	å	Latin small letter a with ring above
230	æ	æ	æ	Latin small letter ae
231	ç	ç	ç	Latin small letter c with cedilla
232	è	è	è	Latin small letter e with grave
233	é	é	é	Latin small letter e with acute
234	ê	ê	ê	Latin small letter e with circumflex
235	ë	ë	ë	Latin small letter e with diaeresis
236	ì	ì	ì	Latin small letter i with grave
237	í	í	í	Latin small letter i with acute
238	î	î	î	Latin small letter i with circumflex
239	ï	ï	ï	Latin small letter i with diaeresis
240	ð	ð	ð	Latin small letter eth
241	ñ	ñ	ñ	Latin small letter n with tilde
242	ò	ò	ò	Latin small letter o with grave
243	ó	ó	ó	Latin small letter o with acute
244	ô	ô	ô	Latin small letter o with circumflex
245	õ	õ	õ	Latin small letter o with tilde
246	ö	ö	ö	Latin small letter o with diaeresis
247	÷	÷	÷	division sign
248	ø	ø	ø	Latin small letter o with stroke
249	ù	ù	ù	Latin small letter u with grave
250	ú	ú	ú	Latin small letter u with acute
251	û	û	û	Latin small letter with circumflex
252	ü	ü	ü	Latin small letter u with diaeresis
253	ý	ý	ý	Latin small letter y with acute
254	þ	þ	þ	Latin small letter thorn
255	ÿ	ÿ	ÿ	Latin small letter y with diaeresis

The ASCII Character Set

ASCII uses the values from 0 to 31 (and 127) for control characters.

ASCII uses the values from 32 to 126 for letters, digits, and symbols.

ASCII does not use the values from 128 to 255.

The ANSI Character Set (Windows-1252)

ANSI is identical to ASCII for the values from 0 to 127.

ANSI has a proprietary set of characters for the values from 128 to 159.

ANSI is identical to UTF-8 for the values from 160 to 255.

The ISO-8859-1 Character Set

8859-1 is identical to ASCII for the values from 0 to 127.

8859-1 does not use the values from 128 to 159.

8859-1 is identical to UTF-8 for the values from 160 to 255.

The UTF-8 Character Set

UTF-8 is identical to ASCII for the values from 0 to 127.

UTF-8 does not use the values from 128 to 159.

UTF-8 is identical to both ANSI and 8859-1 for the values from 160 to 255.

UTF-8 continues from the value 256 with more than 10 000 different characters.

For a closer look, study our [Complete HTML Character Set Reference](#).

The @charset CSS Rule

You can use the CSS **@charset** rule to specify the character encoding used in a style sheet:

Example

Set the encoding of the style sheet to Unicode UTF-8:

```
@charset "UTF-8";
```

HTML Uniform Resource Locators

A URL can be composed of words (w3schools.com), or an Internet Protocol (IP) address (192.68.20.50).

Most people enter the name when surfing, because names are easier to remember than numbers.

Web browsers request pages from web servers by using a URL.

A Uniform Resource Locator (URL) is used to address a document (or other data) on the web.

A web address like <https://www.w3schools.com/html/default.asp> follows these syntax rules:

scheme://prefix.domain:port/path/filename

Explanation:

- **scheme** - defines the **type** of Internet service (most common is **http or https**)
- **prefix** - defines a domain **prefix** (default for http is **www**)
- **domain** - defines the Internet **domain name** (like w3schools.com)
- **port** - defines the **port number** at the host (default for http is **80**)
- **path** - defines a **path** at the server (If omitted: the root directory of the site)
- **filename** - defines the name of a document or resource

Common URL Schemes

The table below lists some common schemes:

Scheme	Short for	Used for
http	HyperText Transfer Protocol	Common web pages. Not encrypted
https	Secure HyperText Transfer Protocol	Secure web pages. Encrypted
ftp	File Transfer Protocol	Downloading or uploading files
file		A file on your computer

URLs can only be sent over the Internet using the ASCII character-set. If a URL contains characters outside the ASCII set, the URL has to be converted.

URL encoding converts non-ASCII characters into a format that can be transmitted over the Internet.

URL encoding replaces non-ASCII characters with a "%" followed by hexadecimal digits.

URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign, or %20.

Try It Yourself

Hello Günter

If you click "Submit", the browser will URL encode the input before it is sent to the server.
A page at the server will display the received input.

Try some other input and click Submit again.

Your browser will encode input, according to the character-set used in your page.

The default character-set in HTML5 is UTF-8.

Character	From Windows-1252	From UTF-8
€	%80	%E2%82%AC
£	%A3	%C2%A3
©	%A9	%C2%A9
®	%AE	%C2%AE
À	%C0	%C3%80
Á	%C1	%C3%81
Â	%C2	%C3%82
Ã	%C3	%C3%83
Ä	%C4	%C3%84
Å	%C5	%C3%85

For a complete reference of all URL encodings, visit our [URL Encoding Reference](#).

HTML and XHTML

XHTML is HTML written as XML.

What Is XHTML?

- XHTML stands for **E**x~~t~~ensible **H**yper**T**ext **M**arkup **L**anguage
- XHTML is almost identical to HTML
- XHTML is stricter than HTML
- XHTML is HTML defined as an XML application
- XHTML is supported by all major browsers

Why XHTML?

Many pages on the internet contain "bad" HTML.

This HTML code works fine in most browsers (even if it does not follow the HTML rules):

```
<html>
<head>
  <title>This is bad HTML</title>

<body>
  <h1>Bad HTML
  <p>This is a paragraph
</body>
```

Today's market consists of different browser technologies. Some browsers run on computers, and some browsers run on mobile phones or other small devices. Smaller devices often lack the resources or power to interpret "bad" markup.

XML is a markup language where documents must be marked up correctly (be "well-formed").

If you want to study XML, please read our [XML tutorial](#).

XHTML was developed by combining the strengths of HTML and XML.

XHTML is HTML redesigned as XML.

The Most Important Differences from HTML:

Document Structure

- XHTML DOCTYPE is **mandatory**
- The `xmlns` attribute in `<html>` is **mandatory**
- `<html>`, `<head>`, `<title>`, and `<body>` are **mandatory**

XHTML Elements

- XHTML elements must be **properly nested**
- XHTML elements must always be **closed**
- XHTML elements must be in **lowercase**
- XHTML documents must have **one root element**

XHTML Attributes

- Attribute names must be in **lower case**
- Attribute values must be **quoted**
- Attribute minimization is **forbidden**

<!DOCTYPE> Is Mandatory

An XHTML document must have an XHTML DOCTYPE declaration.

A complete list of all the [XHTML DocTypes](#) is found in our HTML Tags Reference.

The <html>, <head>, <title>, and <body> elements must also be present, and the xmlns attribute in <html> must specify the xml namespace for the document.

This example shows an XHTML document with a minimum of required tags:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Title of document</title>
</head>

<body>
  some content
</body>

</html>
```

XHTML Elements Must Be Properly Nested

In HTML, some elements can be improperly nested within each other, like this:

```
<b><i>This text is bold and italic</b></i>
```

In XHTML, all elements must be properly nested within each other, like this:

```
<b><i>This text is bold and italic</i></b>
```

XHTML Elements Must Always Be Closed

This is wrong:

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

This is correct:

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

Empty Elements Must Also Be Closed

This is wrong:

```
A break: <br>  
A horizontal rule: <hr>  
An image: 
```

This is correct:

```
A break: <br />
A horizontal rule: <hr />
An image: 
```

XHTML Elements Must Be In Lower Case

This is wrong:

```
<BODY>
<P>This is a paragraph</P>
</BODY>
```

This is correct:

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

XHTML Attribute Names Must Be In Lower Case

This is wrong:

```
<table WIDTH="100%">
```

This is correct:

```
<table width="100%">
```

Attribute Values Must Be Quoted

This is wrong:

```
<table width=100%>
```

This is correct:

```
<table width="100%">
```

Attribute Minimization Is Forbidden

Wrong:

```
<input type="checkbox" name="vehicle" value="car" checked />
```

Correct:

```
<input type="checkbox" name="vehicle" value="car" checked="checked" />
```

Wrong:

```
<input type="text" name="lastname" disabled />
```

Correct:

```
<input type="text" name="lastname" disabled="disabled" />
```

How to Convert from HTML to XHTML

1. Add an XHTML <!DOCTYPE> to the first line of every page
2. Add an xmlns attribute to the html element of every page
3. Change all element names to lowercase
4. Close all empty elements
5. Change all attribute names to lowercase
6. Quote all attribute values

Validate HTML With The W3C Validator

Put your web address in the box below:

https://www.w3schools.com/html/html_validate.html

Validate the page

HTML Forms

HTML Form Example

First name:

Last name:

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 can be displayed in several ways, depending on the **type** attribute.

Here are some examples:

Type	Description
<code><input type="text"></code>	Defines a one-line text input field
<code><input type="radio"></code>	Defines a radio button (for selecting one of many choices)
<code><input type="submit"></code>	Defines a submit button (for submitting the form)

You will learn a lot more about input types later in this tutorial.

Text Input

`<input type="text">` defines a one-line input field for **text input**:

Example

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

This is how it will look like in a browser:

First name: Last name:

Note: The form itself is not visible. Also note that the default width of a text field is 20 characters.

Radio Button Input

`<input type="radio">` defines a **radio button**.

Radio buttons let a user select ONE of a limited number of choices:

Example

```
<form>
  <input type="radio" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other
</form>
```

This is how the HTML code above will be displayed in a browser:

Male Female Other

The Submit Button

`<input type="submit">` defines a button for **submitting** the 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">
    First name:<br>
    <input type="text" name="firstname" value="Mickey"><br>
    Last name:<br>
    <input type="text" name="lastname" value="Mouse"><br><br>
    <input type="submit" value="Submit">
</form>
```

This is how the HTML code above will be displayed in a browser:

First name: Last name:

The Action Attribute

The **action** attribute defines the action to be performed when the form is submitted.

Normally, the form data is sent to a web 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

```
<form action="/action_page.php" target="_blank">
```

Other legal values are "`_parent`", "`_top`", or a name representing the name of an iframe.

The Method Attribute

The `method` attribute specifies the HTTP method (**GET** or **POST**) to be used when submitting the form data:

Example

```
<form action="/action_page.php" method="get">
```

or:

Example

```
<form action="/action_page.php" method="post">
```

When to Use GET?

The default method when submitting form data is GET.

However, when GET is used, the submitted form data will be **visible in the page address field**:

```
/action_page.php?firstname=Mickey&lastname=Mouse
```

Notes on GET:

- Appends form-data into the URL in name/value pairs
- The length of a URL is limited (about 3000 characters)
- Never use GET to send sensitive data! (will be visible in the URL)
- Useful for form submissions where a user wants to bookmark the result
- GET is better for non-secure data, like query strings in Google

When to Use POST?

Always use POST if the form data contains sensitive or personal information. The POST method does not display the submitted form data in the page address field.

Notes on POST:

- POST has no size limitations, and can be used to send large amounts of data.
- Form submissions with POST cannot be bookmarked

The Name Attribute

Each input field must have a `name` attribute to be submitted.

If the `name` attribute is omitted, the data of that input field will not be sent at all.

This example will only submit the "Last name" input field:

Example

```
<form action="/action_page.php">
    First name:<br>
    <input type="text" value="Mickey"><br>
    Last name:<br>
    <input type="text" name="lastname" value="Mouse"><br><br>
    <input type="submit" value="Submit">
</form>
```

Grouping Form Data with <fieldset>

The `<fieldset>` element is used to group related data in a form.

The `<legend>` element defines a caption for the `<fieldset>` element.

Example

```
<form action="/action_page.php">
<fieldset>
  <legend>Personal information:</legend>
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
</fieldset>
</form>
```

This is how the HTML code above will be displayed in a browser:

Personal information:

First name: <input type="text" value="Mickey"/>	Last name: <input type="text" value="Mouse"/>	<input type="button" value="Submit"/>
---	---	---------------------------------------

Here is the list of all `<form>` attributes:

Attribute	Description
accept-charset	Specifies the charset used in the submitted form (default: the page charset).
action	Specifies an address (url) where to submit the form (default: the submitting page).
autocomplete	Specifies if the browser should autocomplete the form (default: on).
enctype	Specifies the encoding of the submitted data (default: is url-encoded).
method	Specifies the HTTP method used when submitting the form (default: GET).
name	Specifies a name used to identify the form (for DOM usage: <code>document.forms.name</code>).
novalidate	Specifies that the browser should not validate the form.
target	Specifies the target of the address in the action attribute (default: <code>_self</code>).

You will learn more about the form attributes in the next chapters.

HTML Form Elements

This chapter describes all HTML form elements.

The <input> Element

The most important form element is the `<input>` element.

The `<input>` element can be displayed in several ways, depending on the `type` attribute.

Example

```
<input name="firstname" type="text">
```

If the `type` attribute is omitted, the input field gets the default type: "text".

All the different input types are covered in the next chapter.

The <select> Element

The `<select>` element defines a **drop-down list**:

Example

```
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The `<option>` elements defines an option that can be selected.

By default, the first item in the drop-down list is selected.

To define a pre-selected option, add the `selected` attribute to the option:

Example

```
<option value="fiat" selected>Fiat</option>
```

Visible Values:

Use the `size` attribute to specify the number of visible values:

Example

```
<select name="cars" size="3">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

Allow Multiple Selections:

Use the `multiple` attribute to allow the user to select more than one value:

Example

```
<select name="cars" size="4" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The <textarea> Element

The `<textarea>` element defines a multi-line input field (**a text area**):

Example

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The `rows` attribute specifies the visible number of lines in a text area.

The `cols` attribute specifies the visible width of a text area.

This is how the HTML code above will be displayed in a browser:

The cat was playing in the garden.

You can also define the size of the text area by using CSS:

Example

```
<textarea name="message" style="width:200px; height:600px">  
The cat was playing in the garden.  
</textarea>
```

The <button> Element

The `<button>` element defines a clickable **button**:

Example

```
<button type="button" onclick="alert('Hello World!')>Click Me!</button>
```

This is how the HTML code above will be displayed in a browser:

Click Me!

Note: Always specify the **type** attribute for the button element. Different browsers may use different default types for the button element.

HTML5 Form Elements

HTML5 added the following form elements:

- `<datalist>`
- `<output>`

Note: Browsers do not display unknown elements. New elements that are not supported in older browsers will not "destroy" your web page.

HTML5 `<datalist>` Element

The `<datalist>` element specifies a list of pre-defined options for an `<input>` element.

Users will see a drop-down list of the pre-defined options as they input data.

The `list` attribute of the `<input>` element, must refer to the `id` attribute of the `<datalist>` element.

Example



```
<form action="/action_page.php">
  <input list="browsers">
  <datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
  </datalist>
</form>
```

The `<output>` element represents the result of a calculation (like one performed by a script).



Example

Perform a calculation and show the result in an `<output>` element:

```
<form action="/action_page.php"
  oninput="x.value=parseInt(a.value)+parseInt(b.value)">
  0
  <input type="range" id="a" name="a" value="50">
  100 +
  <input type="number" id="b" name="b" value="50">
  =
  <output name="x" for="a b"></output>
  <br><br>
  <input type="submit">
</form>
```

HTML Form Elements

= new in HTML5.

Tag	Description
<u><form></u>	Defines an HTML form for user input
<u><input></u>	Defines an input control
<u><textarea></u>	Defines a multiline input control (text area)
<u><label></u>	Defines a label for an <input> element
<u><fieldset></u>	Groups related elements in a form
<u><legend></u>	Defines a caption for a <fieldset> element
<u><select></u>	Defines a drop-down list
<u><optgroup></u>	Defines a group of related options in a drop-down list
<u><option></u>	Defines an option in a drop-down list
<u><button></u>	Defines a clickable button
<u><datalist></u>	Specifies a list of pre-defined options for input controls
<u><output></u>	Defines the result of a calculation

HTML Input Types

This chapter describes the different input types for the `<input>` element.

Input Type Text

`<input type="text">` defines a **one-line text input field**:

Example

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

This is how the HTML code above will be displayed in a browser:

--	--

Input Type Password

`<input type="password">` defines a **password field**:

Example

```
<form>
  User name:<br>
  <input type="text" name="username"><br>
  User password:<br>
  <input type="password" name="psw">
</form>
```

This is how the HTML code above will be displayed in a browser:

--	--

The characters in a password field are masked (shown as asterisks or circles).

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">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
</form>
```

This is how the HTML code above will be displayed in a browser:

First name: Last name:

If you omit the submit button's value attribute, the button will get a default text:

Example

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="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">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
  <input type="reset">
</form>
```

This is how the HTML code above will be displayed in a browser:

First name: Mickey Last name: Mouse Submit
リセット

If you change the input values and then click the "Reset" button, the form-data will be reset to the default values.

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" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other
</form>
```

This is how the HTML code above will be displayed in a browser:



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" name="vehicle1" value="Bike"> I have a bike<br>
  <input type="checkbox" name="vehicle2" value="Car"> I have a car
</form>
```

This is how the HTML code above will be displayed in a browser:



Input Type Button

`<input type="button">` defines a **button**:

Example

```
<input type="button" onclick="alert('Hello World!')" value="Click Me!">
```

This is how the HTML code above will be displayed in a browser:

Click Me!

HTML5 Input Types

HTML5 added several new input types:

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

New input types that are not supported by older web browsers, will behave as `<input type="text">`.

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>
  Select your favorite color:
  <input type="color" name="favcolor">
</form>
```

Input Type Date

The `<input type="date">` is used for input fields that should contain a date.

Depending on browser support, a date picker can show up in the input field.

Example



```
<form>
  Birthday:
  <input type="date" name="bday">
</form>
```

You can also use the `min` and `max` attributes to add restrictions to dates:

Example



```
<form>
  Enter a date before 1980-01-01:
  <input type="date" name="bday" max="1979-12-31"><br>
  Enter a date after 2000-01-01:
  <input type="date" name="bday" min="2000-01-02"><br>
</form>
```

Input Type Datetime-local

The `<input type="datetime-local">` specifies a date and time input field, with no time zone.

Depending on browser support, a date picker can show up in the input field.

Example



```
<form>
  Birthday (date and time):
  <input type="datetime-local" name="bdytime">
</form>
```

Input Type Email

The `<input type="email">` is used for input fields that should contain an e-mail address. Depending on browser support, the e-mail address can be automatically validated when submitted.

Some smartphones recognize the email type, and adds ".com" to the keyboard to match email input.

Example

```
<form>
  E-mail:
  <input type="email" name="email">
</form>
```



Input Type File

The `<input type="file">` defines a file-select field and a "Browse" button for file uploads.

Example

```
<form>
  Select a file: <input type="file" name="myFile">
</form>
```



Input Type Month

The `<input type="month">` allows the user to select a month and year.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  Birthday (month and year):
  <input type="month" name="bdaymonth">
</form>
```



Input Type Number

The `<input type="number">` defines a **numeric** input field.

You can also set restrictions on what numbers are accepted.

The following example displays a numeric input field, where you can enter a value from 1 to 5:



Example

```
<form>
  Quantity (between 1 and 5):
  <input type="number" name="quantity" min="1" max="5">
</form>
```

Input Restrictions

Here is a list of some common input restrictions (some are new in HTML5):

Attribute	Description
disabled	Specifies that an input field should be disabled
max	Specifies the maximum value for an input field
maxlength	Specifies the maximum number of character for an input field
min	Specifies the minimum value for an input field
pattern	Specifies a regular expression to check the input value against
readonly	Specifies that an input field is read only (cannot be changed)
required	Specifies that an input field is required (must be filled out)
size	Specifies the width (in characters) of an input field
step	Specifies the legal number intervals for an input field
value	Specifies the default value for an input field

You will learn more about input restrictions in the next chapter.

The following example displays a numeric input field, where you can enter a value from 0 to 100, in steps of 10. The default value is 30:

Example



```
<form>
    Quantity:
    <input type="number" name="points" min="0" max="100" step="10"
    value="30">
</form>
```

Input Type Range

The `<input type="range">` defines a control for entering a number whose exact value is not important (like a slider control). Default range is 0 to 100. However, you can set restrictions on what numbers are accepted with the `min`, `max`, and `step` attributes:

Example



```
<form>
    <input type="range" name="points" min="0" max="10">
</form>
```

Input Type Search

The `<input type="search">` is used for search fields (a search field behaves like a regular text field).

Example



```
<form>
    Search Google:
    <input type="search" name="googlesearch">
</form>
```

Input Type Tel

The `<input type="tel">` is used for input fields that should contain a telephone number.

Note: The tel type is currently only supported in Safari 8.



Example

```
<form>
  Telephone:
  <input type="tel" name="usrtel">
</form>
```

Input Type Time

The `<input type="time">` allows the user to select a time (no time zone).

Depending on browser support, a time picker can show up in the input field.



Example

```
<form>
  Select a time:
  <input type="time" name="usr_time">
</form>
```

Input Type Url

The `<input type="url">` is used for input fields that should contain a URL address.

Depending on browser support, the url field can be automatically validated when submitted.

Some smartphones recognize the url type, and adds ".com" to the keyboard to match url input.



Example

```
<form>
  Add your homepage:
  <input type="url" name="homepage">
</form>
```

Input Type Week

The `<input type="week">` allows the user to select a week and year.

Depending on browser support, a date picker can show up in the input field.



Example

```
<form>
  Select a week:
  <input type="week" name="week_year">
</form>
```

HTML Input Type Attribute

Tag	Description
<code><input type=""></code>	Specifies the input type to display

HTML Input Attributes

The value Attribute

The `value` attribute specifies the initial value for an input field:

Example

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John">
</form>
```

The readonly Attribute

The `readonly` attribute specifies that the input field is read only (cannot be changed):

Example

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" readonly>
</form>
```

The disabled Attribute

The `disabled` attribute specifies that the input field is disabled.

A disabled input field is unusable and un-clickable, and its value will not be sent when submitting the form:

Example

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" disabled>
</form>
```

The size Attribute

The `size` attribute specifies the size (in characters) for the input field:

Example

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" size="40">
</form>
```

The maxlength Attribute

The `maxlength` attribute specifies the maximum allowed length for the input field:

Example

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" maxlength="10">
</form>
```

With a `maxlength` attribute, the input field will not accept more than the allowed number of characters.

The `maxlength` attribute does not provide any feedback. If you want to alert the user, you must write JavaScript code.

Note: Input restrictions are not foolproof, and JavaScript provides many ways to add illegal input. To safely restrict input, it must be checked by the receiver (the server) as well!

HTML5 Attributes

HTML5 added the following attributes for `<input>`:

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

and the following attributes for `<form>`:

- autocomplete
- novalidate

The autocomplete Attribute

The `autocomplete` attribute specifies whether a form or input field should have autocomplete on or off.

When autocomplete is on, the browser automatically completes the input values based on values that the user has entered before.

Tip: It is possible to have autocomplete "on" for the form, and "off" for specific input fields, or vice versa.

The `autocomplete` attribute works with `<form>` and the following `<input>` types: text, search, url, tel, email, password, datepickers, range, and color.

Example

An HTML form with autocomplete on (and off for one input field):

```
<form action="/action_page.php" autocomplete="on">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  E-mail: <input type="email" name="email" autocomplete="off"><br>
  <input type="submit">
</form>
```



Tip: In some browsers you may need to activate the autocomplete function for this to work.

The novalidate Attribute

The `novalidate` attribute is a `<form>` attribute.

When present, novalidate specifies that the form data should not be validated when submitted.

Example

Indicates that the form is not to be validated on submit:

```
<form action="/action_page.php" novalidate>
  E-mail: <input type="email" name="user_email">
  <input type="submit">
</form>
```



The autofocus Attribute

The `autofocus` attribute specifies that the input field should automatically get focus when the page loads.

Example

Let the "First name" input field automatically get focus when the page loads:



```
First name:<input type="text" name="fname" autofocus>
```

The form Attribute

The `form` attribute specifies one or more forms an `<input>` element belongs to.

Tip: To refer to more than one form, use a space-separated list of form ids.

Example

An input field located outside the HTML form (but still a part of the form):



```
<form action="/action_page.php" id="form1">
  First name: <input type="text" name="fname"><br>
  <input type="submit" value="Submit">
</form>

Last name: <input type="text" name="lname" form="form1">
```

The formaction Attribute

The `formaction` attribute specifies the URL of a file that will process the input control when the form is submitted.

The `formaction` attribute overrides the `action` attribute of the `<form>` element.

The `formaction` attribute is used with `type="submit"` and `type="image"`.



Example

An HTML form with two submit buttons, with different actions:

```
<form action="/action_page.php">
    First name: <input type="text" name="fname"><br>
    Last name: <input type="text" name="lname"><br>
    <input type="submit" value="Submit"><br>
    <input type="submit" formaction="/action_page2.php"
          value="Submit as admin">
</form>
```

The formenctype Attribute

The `formenctype` attribute specifies how the form data should be encoded when submitted (only for forms with `method="post"`).

The `formenctype` attribute overrides the `enctype` attribute of the `<form>` element.

The `formenctype` attribute is used with `type="submit"` and `type="image"`.



Example

Send form-data that is default encoded (the first submit button), and encoded as "multipart/form-data" (the second submit button):

```
<form action="/action_page_binary.asp" method="post">
    First name: <input type="text" name="fname"><br>
    <input type="submit" value="Submit">
    <input type="submit" formenctype="multipart/form-data"
          value="Submit as Multipart/form-data">
</form>
```

The formmethod Attribute

The `formmethod` attribute defines the HTTP method for sending form-data to the action URL.

The `formmethod` attribute overrides the method attribute of the `<form>` element.

The `formmethod` attribute can be used with `type="submit"` and `type="image"`.

Example

The second submit button overrides the HTTP method of the form:

```
<form action="/action_page.php" method="get">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
  <input type="submit" formmethod="post" value="Submit using POST">
</form>
```



The formnovalidate Attribute

The `formnovalidate` attribute overrides the novalidate attribute of the `<form>` element.

The `formnovalidate` attribute can be used with `type="submit"`.

Example

A form with two submit buttons (with and without validation):

```
<form action="/action_page.php">
  E-mail: <input type="email" name="userid"><br>
  <input type="submit" value="Submit"><br>
  <input type="submit" formnovalidate value="Submit without validation">
</form>
```



The formtarget Attribute

The `formtarget` attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.

The `formtarget` attribute overrides the target attribute of the `<form>` element.

The `formtarget` attribute can be used with `type="submit"` and `type="image"`.



Example

A form with two submit buttons, with different target windows:

```
<form action="/action_page.php">
    First name: <input type="text" name="fname"><br>
    Last name: <input type="text" name="lname"><br>
    <input type="submit" value="Submit as normal">
    <input type="submit" formtarget="_blank"
          value="Submit to a new window">
</form>
```

The height and width Attributes

The `height` and `width` attributes specify the height and width of an `<input type="image">` element.

Always specify the size of images. If the browser does not know the size, the page will flicker while images load.



Example

Define an image as the submit button, with height and width attributes:

```
<input type="image" src="img_submit.gif" alt="Submit" width="48"
      height="48">
```

The list Attribute

The `list` attribute refers to a `<datalist>` element that contains pre-defined options for an `<input>` element.



Example

An `<input>` element with pre-defined values in a `<datalist>`:

```
<input list="browsers">

<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

The min and max Attributes

The `min` and `max` attributes specify the minimum and maximum values for an `<input>` element.

The `min` and `max` attributes work with the following input types: number, range, date, datetime-local, month, time and week.



Example

`<input>` elements with min and max values:

```
Enter a date before 1980-01-01:
<input type="date" name="bday" max="1979-12-31">

Enter a date after 2000-01-01:
<input type="date" name="bday" min="2000-01-02">

Quantity (between 1 and 5):
<input type="number" name="quantity" min="1" max="5">
```

The multiple Attribute

The `multiple` attribute specifies that the user is allowed to enter more than one value in the `<input>` element.

The `multiple` attribute works with the following input types: email, and file.



Example

A file upload field that accepts multiple values:

```
Select images: <input type="file" name="img" multiple>
```

The pattern Attribute

The `pattern` attribute specifies a regular expression that the `<input>` element's value is checked against.

The `pattern` attribute works with the following input types: text, search, url, tel, email, and password.

Tip: Use the global `title` attribute to describe the pattern to help the user.

Tip: Learn more about [regular expressions](#) in our JavaScript tutorial.



Example

An input field that can contain only three letters (no numbers or special characters):

```
Country code: <input type="text" name="country_code" pattern="[A-Za-z]{3}" title="Three letter country code">
```

The placeholder Attribute

The `placeholder` attribute specifies a hint that describes the expected value of an input field (a sample value or a short description of the format).

The hint is displayed in the input field before the user enters a value.

The `placeholder` attribute works with the following input types: text, search, url, tel, email, and password.

Example

An input field with a placeholder text:

```
<input type="text" name="fname" placeholder="First name">
```



The required Attribute

The `required` attribute specifies that an input field must be filled out before submitting the form.

The `required` attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

Example

A required input field:

```
Username: <input type="text" name="username" required>
```



The step Attribute

The `step` attribute specifies the legal number intervals for an `<input>` element.

Example: if `step="3"`, legal numbers could be -3, 0, 3, 6, etc.

Tip: The `step` attribute can be used together with the `max` and `min` attributes to create a range of legal values.

The `step` attribute works with the following input types: number, range, date, datetime-local, month, time and week.

Example

An input field with a specified legal number intervals:



```
<input type="number" name="points" step="3">
```

HTML Form and Input Elements

Tag	Description
<code><form></code>	Defines an HTML form for user input
<code><input></code>	Defines an input control

HTML5 Introduction

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>`.

In the next chapter, [HTML5 Support](#), you will learn how to "teach" older browsers to handle "unknown" (new) HTML elements.

New HTML5 API's (Application Programming Interfaces)

The most interesting new API's 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.

Removed Elements in HTML5

The following HTML4 elements have been removed in HTML5:

Removed Element	Use Instead
<acronym>	<abbr>
<applet>	<object>
<basefont>	CSS
<big>	CSS
<center>	CSS
<dir>	
	CSS
<frame>	
<frameset>	
<noframes>	
<strike>	CSS, <s>, or
<tt>	CSS

In the chapter [HTML5 Migration](#), you will learn how to easily migrate from HTML4 to HTML5.

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.

The WHATWG HTML5 Living Standard was published in 2012, and is continuously updated.

W3C wanted to develop a definitive HTML5 and XHTML standard.

The W3C HTML5 Recommendation was released 28 October 2014.

HTML5 Browser Support

You can teach older browsers to handle HTML5 correctly.

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.

Define Semantic Elements as Block Elements

HTML5 defines eight new **semantic** elements. All these are **block-level** elements.

To secure correct behavior in older browsers, you can set the CSS **display** property for these HTML elements to **block**:

```
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:

Example

```
<!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>
  
</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
<code><article></code>	Defines an article in a document
<code><aside></code>	Defines content aside from the page content
<code><bdi></code>	Isolates a part of text that might be formatted in a different direction from other text outside it
<code><details></code>	Defines additional details that the user can view or hide
<code><dialog></code>	Defines a dialog box or window
<code><figcaption></code>	Defines a caption for a <code><figure></code> element
<code><figure></code>	Defines self-contained content
<code><footer></code>	Defines a footer for a document or section
<code><header></code>	Defines a header for a document or section
<code><main></code>	Defines the main content of a document
<code><mark></code>	Defines marked/highlighted text
<code><meter></code>	Defines a scalar measurement within a known range (a gauge)
<code><nav></code>	Defines navigation links
<code><progress></code>	Represents the progress of a task
<code><rp></code>	Defines what to show in browsers that do not support ruby annotations
<code><rt></code>	Defines an explanation/pronunciation of characters (for East Asian typography)
<code><ruby></code>	Defines a ruby annotation (for East Asian typography)
<code><section></code>	Defines a section in a document
<code><summary></code>	Defines a visible heading for a <code><details></code> element
<code><time></code>	Defines a date/time
<code><wbr></code>	Defines a possible line-break

Read more about [HTML5 Semantics](#).

New Form Elements

Tag	Description
<code><datalist></code>	Specifies a list of pre-defined options for input controls
<code><output></code>	Defines the result of a calculation

Read all about old and new form elements in [HTML Form Elements](#).

New Input Types

New Input Types	New Input Attributes
<ul style="list-style-type: none">• color• date• datetime• datetime-local• email• month• number• range• search• tel• time• url• week	<ul style="list-style-type: none">• autocomplete• autofocus• form• formaction• formenctype• formmethod• formnovalidate• formtarget• height and width• list• min and max• multiple• pattern (regexp)• placeholder• required• step

Learn all about old and new input types in [HTML Input Types](#).

Learn all about input attributes in [HTML Input Attributes](#).

HTML5 - New Attribute Syntax

HTML5 allows four different syntaxes for attributes.

This example demonstrates the different syntaxes used in an `<input>` tag:

Type	Example
Empty	<code><input type="text" value="John" disabled></code>
Unquoted	<code><input type="text" value=John></code>
Double-quoted	<code><input type="text" value="John Doe"></code>
Single-quoted	<code><input type="text" value='John Doe'></code>

In HTML5, all four syntaxes may be used, depending on what is needed for the attribute.

HTML5 Graphics

Tag	Description
<code><canvas></code>	Draw graphics, on the fly, via scripting (usually JavaScript)
<code><svg></code>	Draw scalable vector graphics

Read more about [HTML5 Canvas](#).

Read more about [HTML5 SVG](#).

New Media Elements

Tag	Description
<code><audio></code>	Defines sound content
<code><embed></code>	Defines a container for an external (non-HTML) application
<code><source></code>	Defines multiple media resources for media elements (<code><video></code> and <code><audio></code>)
<code><track></code>	Defines text tracks for media elements (<code><video></code> and <code><audio></code>)
<code><video></code>	Defines video or movie

Read more about [HTML5 Video](#).

HTML5 Semantic Elements

Semantics is the study of the meanings of words and phrases in a language.
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 `` - Tells nothing about its content.
Examples of **semantic** elements: `<form>`, `<table>`, and `<article>` - Clearly defines its content.

Browser Support

				
Yes	Yes	Yes	Yes	Yes

HTML5 semantic elements are supported in all modern browsers.

In addition, you can "teach" older browsers how to handle "unknown elements".

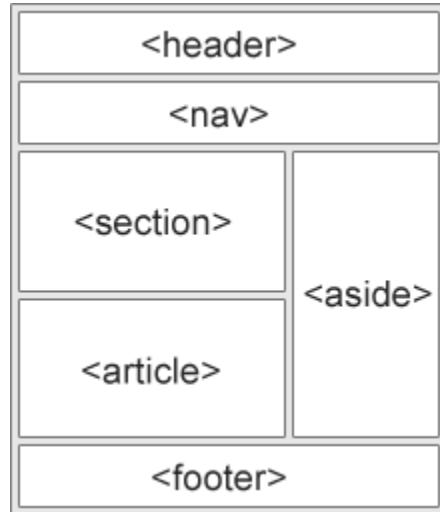
Read about it in [HTML5 Browser Support](#).

New Semantic Elements in HTML5

Many web sites contain HTML code like: `<div id="nav">` `<div class="header">` `<div id="footer">` to indicate navigation, header, and footer.

HTML5 offers new semantic elements to define different parts of a web page:

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



HTML5 <section> Element

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

According to W3C's HTML5 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>
```

HTML5 <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>`.

HTML5 <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>
```

HTML5 <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>
```

HTML5 <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>
```

HTML5 <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>
```

HTML5 <figure> and <figcaption> Elements

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

In HTML5, an image and a caption can be grouped together in a `<figure>` element:

Example

```
<figure>
  
  <figcaption>Fig1. - Trulli, Puglia, Italy.</figcaption>
</figure>
```

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

Why Semantic Elements?

With HTML4, developers used their own id/class names to style elements: header, top, bottom, footer, menu, navigation, main, container, content, article, sidebar, topnav, etc.

This made it impossible for search engines to identify the correct web page content.

With the new HTML5 elements (`<header>` `<footer>` `<nav>` `<section>` `<article>`), this will become easier.

According to the W3C, a Semantic Web: "Allows data to be shared and reused across applications, enterprises, and communities."

Semantic Elements in HTML5

Below is an alphabetical list of the new semantic elements in HTML5.

The links go to our complete [HTML5 Reference](#).

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

HTML5 Migration

Migration from HTML4 to HTML5

This chapter is entirely about how to **migrate** from **HTML4** to **HTML5**.

This chapter demonstrates how to convert an HTML4 page into an HTML5 page, without destroying anything of the original content or structure.

You can migrate from XHTML to HTML5, using the same recipe.

Typical HTML4 Typical HTML5

```
<div id="header">    <header>
<div id="menu">      <nav>
<div id="content">   <section>
<div class="article"> <article>
<div id="footer">    <footer>
```

A Typical HTML4 Page

Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>HTML4</title>
<style>
body {
    font-family: Verdana, sans-serif;
    font-size: 0.9em;
}

div#header, div#footer {
    padding: 10px;
    color: white;
    background-color: black;
}

div#content {
    margin: 5px;
```

```
padding: 10px;
background-color: lightgrey;
}

div.article {
margin: 5px;
padding: 10px;
background-color: white;
}

div#menu ul {
padding: 0;
}

div#menu ul li {
display: inline;
margin: 5px;
}

</style>
</head>
<body>

<div id="header">
  <h1>Monday Times</h1>
</div>

<div id="menu">
  <ul>
    <li>News</li>
    <li>Sports</li>
    <li>Weather</li>
  </ul>
</div>

<div id="content">
  <h2>News Section</h2>
  <div class="article">
    <h2>News Article</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Pellentesque in porta lorem. Morbi condimentum est nibh, et consectetur  
tortor feugiat at.</p>
  </div>
  <div class="article">
```

```
<h2>News Article</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Pellentesque in porta lorem. Morbi condimentum est nibh, et consectetur
tortor feugiat at.</p>
</div>
</div>

<div id="footer">
<p>&copy; 2016 Monday Times. All rights reserved.</p>
</div>

</body>
</html>
```

Change to HTML5 Doctype

Change the **doctype**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

to the HTML5 doctype:

Example

```
<!DOCTYPE html>
```

Change to HTML5 Encoding

Change the **encoding** information:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

to HTML5 encoding:

Example

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

Add The HTML5Shiv

The new HTML5 semantic elements are supported in all modern browsers. In addition, you can "teach" older browsers how to handle "unknown elements".

However, IE8 and earlier, does not allow styling of unknown elements. So, the HTML5Shiv is a JavaScript workaround to enable styling of HTML5 elements in versions of Internet Explorer prior to version 9.

Add the HTML5Shiv:

Example

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

Read more about the **HTML5Shiv** in [HTML5 Browser Support](#).

Change to HTML5 Semantic Elements

The existing CSS contains id's and classes for styling the elements:

```
body {  
    font-family: Verdana, sans-serif;  
    font-size: 0.9em;  
}  
  
div#header, div#footer {  
    padding: 10px;  
    color: white;  
    background-color: black;  
}  
  
div#content {  
    margin: 5px;  
    padding: 10px;  
    background-color: lightgrey;  
}  
  
div.article {  
    margin: 5px;  
    padding: 10px;  
    background-color: white;  
}  
  
div#menu ul {  
    padding: 0;  
}  
  
div#menu ul li {  
    display: inline;  
    margin: 5px;  
}
```

Replace with equal CSS styles for HTML5 semantic elements:

```
body {  
    font-family: Verdana, sans-serif;  
    font-size: 0.9em;  
}  
  
header, footer {  
    padding: 10px;  
    color: white;  
    background-color: black;  
}  
  
section {  
    margin: 5px;  
    padding: 10px;  
    background-color: lightgrey;  
}  
  
article {  
    margin: 5px;  
    padding: 10px;  
    background-color: white;  
}  
  
nav ul {  
    padding: 0;  
}  
  
nav ul li {  
    display: inline;  
    margin: 5px;  
}
```

Finally, change the elements to HTML5 semantic elements:

Example

```
<body>

<header>
<h1>Monday Times</h1>
</header>

<nav>
<ul>
<li>News</li>
<li>Sports</li>
<li>Weather</li>
</ul>
</nav>

<section>
<h2>News Section</h2>
<article>
<h2>News Article</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in
porta lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.
</p>
</article>
<article>
<h2>News Article</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in
porta lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.
</p>
</article>
</section>

<footer>
<p>&copy; 2014 Monday Times. All rights reserved.</p>
</footer>

</body>
```

The Difference Between <article> <section> and <div>

There is a confusing (lack of) difference in the HTML5 standard, between `<article>` `<section>` and `<div>`.

In the HTML5 standard, the `<section>` element is defined as a block of related elements.
The `<article>` element is defined as a complete, self-contained block of related elements.
The `<div>` element is defined as a block of children elements.

How to interpret that?

In the example above, we have used `<section>` as a container for related `<articles>`.

But, we could have used `<article>` as a container for articles as well.

Here are some different examples:

`<article> in <article>:`

```
<article>

<h2>Famous Cities</h2>

<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>

</article>
```

<div> in <article>:

```
<article>

<h2>Famous Cities</h2>

<div class="city">
<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>
</div>

<div class="city">
<h2>Paris</h2>
<p>Paris is the capital and most populous city of France.</p>
</div>

<div class="city">
<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>
</div>

</article>
```

<div> in <section> in <article>:

```
<article>

<section>
<h2>Famous Cities</h2>

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

```
<div class="city">
<h2>Paris</h2>
<p>Paris is the capital and most populous city of France.</p>
</div>

<div class="city">
<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>
</div>
</section>

<section>
<h2>Famous Countries</h2>

<div class="country">
<h2>England</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>
</div>

<div class="country">
<h2>France</h2>
<p>Paris is the capital and most populous city of France.</p>
</div>

<div class="country">
<h2>Japan</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>
</div>
</section>

</article>
```

HTML5 Style Guide and Coding Conventions

HTML Coding Conventions

Web developers are often uncertain about the coding style and syntax to use in HTML.

Between 2000 and 2010, many web developers converted from HTML to XHTML.

With XHTML, developers were forced to write valid and "well-formed" code.

HTML5 is a bit more sloppy when it comes to code validation.

Be Smart and Future Proof

A consistent use of style makes it easier for others to understand your HTML.

In the future, programs like XML readers may want to read your HTML.

Using a well-formed—"close to XHTML" syntax can be smart.

Always keep your code tidy, clean and well-formed.

Use Correct Document Type

Always declare the document type as the first line in your document:

```
<!DOCTYPE html>
```

If you want consistency with lower case tags, you can use:

```
<!doctype html>
```

Use Lower Case Element Names

HTML5 allows mixing uppercase and lowercase letters in element names.

We recommend using lowercase element names because:

- Mixing uppercase and lowercase names is bad
- Developers normally use lowercase names (as in XHTML)
- Lowercase look cleaner
- Lowercase are easier to write

Bad:

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

Very Bad:

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

Good:

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

Close All HTML Elements

In HTML5, you don't have to close all elements (for example the `<p>` element). We recommend closing all HTML elements.

Bad:

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

Good:

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

Close Empty HTML Elements

In HTML5, it is optional to close empty elements.

Allowed:

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

Also Allowed:

```
<meta charset="utf-8" />
```

However, the closing slash (/) is REQUIRED in XHTML and XML.

If you expect XML software to access your page, it is a good idea to keep the closing slash!

Use Lower Case Attribute Names

HTML5 allows mixing uppercase and lowercase letters in attribute names.

We recommend using lowercase attribute names because:

- Mixing uppercase and lowercase names is bad
- Developers normally use lowercase names (as in XHTML)
- Lowercase look cleaner
- Lowercase are easier to write

Bad:

```
<div CLASS="menu">
```

Good:

```
<div class="menu">
```

Quote Attribute Values

HTML5 allows attribute values without quotes.

We recommend quoting attribute values because:

- Mixing uppercase and lowercase values is bad
- Quoted values are easier to read
- You MUST use quotes if the value contains spaces

Very bad:

This will not work, because the value contains spaces:

```
<table class=table striped>
```

Bad:

```
<table class=striped>
```

Good:

```
<table class="striped">
```

Image Attributes

Always add the `alt` attribute to images. This attribute is important when the image for some reason cannot be displayed. Also, always define image width and height. It reduces flickering because the browser can reserve space for the image before loading.

Bad:

```

```

Good:

```

```

Spaces and Equal Signs

HTML5 allows spaces around equal signs. But space-less is easier to read and groups entities better together.

Bad:

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

Good:

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

Avoid Long Code Lines

When using an HTML editor, it is inconvenient to scroll right and left to read the HTML code. Try to avoid code lines longer than 80 characters.

Blank Lines and Indentation

Do not add blank lines without a reason.

For readability, add blank lines to separate large or logical code blocks.

For readability, add two spaces of indentation. Do not use the tab key.

Do not use unnecessary blank lines and indentation. It is not necessary to indent every element:

Unnecessary:

```
<body>

<h1>Famous Cities</h1>

<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.
    It is the seat of the Japanese government and the Imperial Palace,
    and the home of the Japanese Imperial Family.
</p>

</body>
```

Better:

```
<body>

<h1>Famous Cities</h1>

<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.
    It is the seat of the Japanese government and the Imperial Palace,
    and the home of the Japanese Imperial Family.</p>

</body>
```

Table Example:

```
<table>
  <tr>
    <th>Name</th>
    <th>Description</th>
  </tr>
  <tr>
    <td>A</td>
    <td>Description of A</td>
  </tr>
  <tr>
    <td>B</td>
    <td>Description of B</td>
  </tr>
</table>
```

List Example:

```
<ol>
  <li>London</li>
  <li>Paris</li>
  <li>Tokyo</li>
</ol>
```

Omitting <html> and <body>?

In HTML5, the `<html>` tag and the `<body>` tag can be omitted.

The following code will validate as HTML5:

Example

```
<!DOCTYPE html>
<head>
  <title>Page Title</title>
</head>

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

However, we do not recommend omitting the `<html>` and the `<body>` tag.

The `<html>` element is the document root. It is the recommended place for specifying the page language:

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

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

Omitting `<html>` or `<body>` can crash DOM and XML software.

Omitting `<body>` can produce errors in older browsers (IE9).

Omitting <head>?

In HTML5, the <head> tag can also be omitted.

By default, browsers will add all elements before <body> to a default <head> element.

You can reduce the complexity of HTML by omitting the <head> tag:

Example

```
<!DOCTYPE html>
<html>
<title>Page Title</title>

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

</html>
```

However, we do not recommend omitting the <head> tag.

Omitting tags is unfamiliar to web developers. It needs time to be established as a guideline.

Meta Data

The <title> element is required in HTML5. Make the title as meaningful as possible:

```
<title>HTML5 Syntax and Coding Style</title>
```

To ensure proper interpretation and correct search engine indexing, both the language and the character encoding should be defined as early as possible in a document:

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>HTML5 Syntax and Coding Style</title>
</head>
```

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the `<meta>` tag.

The viewport is the user's visible area of a web page. It varies with the device, and will be smaller on a mobile phone than on a computer screen.

You should include the following `<meta>` viewport element in all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

A `<meta>` viewport element gives the browser instructions on how to control the page's dimensions and scaling.

The `width=device-width` part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The `initial-scale=1.0` part sets the initial zoom level when the page is first loaded by the browser.

Here is an example of a web page *without* the viewport meta tag, and the same web page *with* the viewport meta tag:

Tip: If you are browsing this page with a phone or a tablet, you can click on the two links below to see the difference.



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming.



Without the

viewport meta tag

With the viewport

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming.

meta tag

HTML Comments

Short comments should be written on one line, like this:

```
<!-- This is a comment -->
```

Comments that spans more than one line, should be written like this:

```
<!--  
 This is a long comment example. This is a long comment example.  
 This is a long comment example. This is a long comment example.  
-->
```

Long comments are easier to observe if they are indented two spaces.

Style Sheets

Use simple syntax for linking to style sheets (the type attribute is not necessary):

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

Short rules can be written compressed, like this:

```
p.intro {font-family: Verdana; font-size: 16em;}
```

Long rules should be written over multiple lines:

```
body {  
    background-color: lightgrey;  
    font-family: "Arial Black", Helvetica, sans-serif;  
    font-size: 16em;  
    color: black;  
}
```

- Place the opening bracket on the same line as the selector
- Use one space before the opening bracket
- Use two spaces of indentation
- Use semicolon after each property-value pair, including the last
- Only use quotes around values if the value contains spaces
- Place the closing bracket on a new line, without leading spaces
- Avoid lines over 80 characters

Loading JavaScript in HTML

Use simple syntax for loading external scripts (the type attribute is not necessary):

```
<script src="myscript.js">
```

Accessing HTML Elements with JavaScript

A consequence of using "untidy" HTML styles can result in JavaScript errors.

These two JavaScript statements will produce different results:

Example

```
var obj = getElementById("Demo")  
  
var obj = getElementById("demo")
```

[Visit the JavaScript Style Guide.](#)

Use Lower Case File Names

Some web servers (Apache, Unix) are case sensitive about file names: "london.jpg" cannot be accessed as "London.jpg".

Other web servers (Microsoft, IIS) are not case sensitive: "london.jpg" can be accessed as "London.jpg" or "london.jpg".

If you use a mix of upper and lower case, you have to be extremely consistent.

If you move from a case insensitive to a case sensitive server, even small errors will break your web!

To avoid these problems, always use lower case file names.

File Extensions

HTML files should have a **.html** or **.htm** extension.

CSS files should have a **.css** extension.

JavaScript files should have a **.js** extension.

Differences Between .htm and .html

There is no difference between the .htm and .html extensions. Both will be treated as HTML by any web browser or web server.

The differences are cultural:

.htm "smells" of early DOS systems where the system limited the extensions to 3 characters.

.html "smells" of Unix operating systems that did not have this limitation.

Technical Differences

When a URL does not specify a filename (like <https://www.w3schools.com/css/>), the server returns a default filename. Common default filenames are index.html, index.htm, default.html and default.htm.

If your server is configured only with "index.html" as default filename, your file must be named "index.html", not "index.htm."

However, servers can be configured with more than one default filename, and normally you can set up as many default filenames as needed.

Anyway, the full extension for HTML files is .html, and there's no reason it should not be used.

HTML Multimedia

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

What is Multimedia?

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 by 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 bandwidths. 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 for iTunes. Plays well on Apple computers, but not in web browsers.
WAV	.wav	WAV. Developed by IBM and Microsoft. Plays well on Windows, Macintosh, and Linux 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. Courtesy of [Big Buck Bunny](#).



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.

Browser Support

The numbers in the table specify the first browser version that fully supports the `<video>` element.

Element					
<code><video></code>	4.0	9.0	3.5	4.0	10.5

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:

Browser	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.

Example: Using JavaScript

Play/Pause Big Small Normal



Video courtesy of [Big Buck Bunny](#).

For a full DOM reference, go to our [HTML5 Audio/Video DOM Reference](#).

HTML5 Video Tags

Tag	Description
<code><video></code>	Defines a video or movie
<code><source></code>	Defines multiple media resources for media elements, such as <code><video></code> and <code><audio></code>
<code><track></code>	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.

Browser Support

The numbers in the table specify the first browser version that fully supports the `<audio>` element.

Element					
<code><audio></code>	4.0	9.0	3.5	4.0	10.5

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>
```

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 - Browser Support

In HTML5, there are 3 supported audio formats: MP3, WAV, and OGG.

The browser support for the different formats is:

Browser	MP3	WAV	OGG
Internet Explorer	YES	NO	NO
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES

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.

For a full DOM reference, go to our [HTML5 Audio/Video DOM Reference](#).

Tag	Description
<code><audio></code>	Defines sound content
<code><source></code>	Defines multiple media resources for media elements, such as <code><video></code> and <code><audio></code>

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>
```

Or images if you like:

Example

```
<object data="audi.jpeg"></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.

Example

```
<embed width="400" height="50" src="bookmark.swf">
```

Note that the `<embed>` element does not have a closing tag. It can not contain alternative text.

The `<embed>` element can also be used to include HTML in HTML:

Example

```
<embed width="100%" height="500px" src="snippet.html">
```

Or images if you like:

Example

```
<embed src="audi.jpeg">
```

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)

Example - Using iFrame (recommended)

```
<iframe width="420" height="315">
  src="https://www.youtube.com/embed/tgbNymZ7vqY"
</iframe>
```

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

```
<iframe width="420" height="315"  
src="https://www.youtube.com/embed/tgbNymZ7vqY?autoplay=1">  
</iframe>
```

YouTube Playlist

A comma separated list of videos to play (in addition to the original URL).

YouTube Loop

Value 0 (default): The video will play only once.

Value 1: The video will loop (forever).

YouTube - Loop

```
<iframe width="420" height="315"  
src="https://www.youtube.com/embed/tgbNymZ7vqY?  
playlist=tgbNymZ7vqY&loop=1">  
</iframe>
```

YouTube Controls

Value 0: Player controls does not display.

Value 1 (default): Player controls display.

YouTube - Controls

```
<iframe width="420" height="315"  
src="https://www.youtube.com/embed/tgbNymZ7vqY?controls=0">  
</iframe>
```

YouTube - Using <object> or <embed>

Note: YouTube `<object>` and `<embed>` were deprecated from January 2015. You should migrate your videos to use `<iframe>` instead.

Example - Using <object> (deprecated)

```
<object width="420" height="315"  
data="https://www.youtube.com/embed/tgbNymZ7vqY">  
</object>
```

Example - Using <embed> (deprecated)

```
<embed width="420" height="315"  
src="https://www.youtube.com/embed/tgbNymZ7vqY">
```

HTML5 Geolocation

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

Try It

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 iPhone.

Browser Support

The numbers in the table specify the first browser version that fully supports Geolocation.

API					
Geolocation	5.0 - 49.0 (http)50.0 (https)	9.0	3.5	5.0	16.0

Note: As of Chrome 50, the Geolocation API will only work on secure contexts such as HTTPS. If your site is hosted on a 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+"'>";  
}
```

[Google Map Script](#) How to show an interactive Google Map with a marker, zoom and drag options.

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 iPhone):

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 the W3Schools image into the rectangle.

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.

Browser Support

The numbers in the table specify the first browser version that fully supports Drag and Drop.

API					
Drag and Drop	4.0	9.0	3.5	6.0	12.0

HTML Drag and Drop Example

The example below is a simple drag and drop example:

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>



</body>
</html>
```

It might seem complicated, but let's 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

More Examples

Drag image back and forth

How to drag (and drop) an image back and forth between two <div> elements:



HTML5 Web Storage

HTML web storage; better than cookies.

What is HTML Web Storage?

With web storage, web applications can store data locally within the user's browser.

Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.

Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

Browser Support

The numbers in the table specify the first browser version that fully supports Web Storage.

API					
Web Storage	4.0	8.0	3.5	4.0	11.5

HTML Web Storage Objects

HTML web storage provides two objects for storing data on the client:

- `window.localStorage` - stores data with no expiration date
- `window.sessionStorage` - stores data for one session (data is lost when the browser tab is closed)

Before using web storage, check browser support for `localStorage` and `sessionStorage`:

```
if (typeof(Storage) !== "undefined") {      // Code for
localStorage/sessionStorage. } else {      // Sorry! No Web Storage support.. }
```

The localStorage Object

The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Example

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML =
localStorage.getItem("lastname");
```

Example explained:

- Create a localStorage name/value pair with name="lastname" and value="Smith"
- Retrieve the value of "lastname" and insert it into the element with id="result"

The example above could also be written like this:

```
// StorelocalStorage.lastname = "Smith"; // Retrieve
document.getElementById("result").innerHTML = localStorage.lastname;
```

The syntax for removing the "lastname" localStorage item is as follows:

```
localStorage.removeItem("lastname");
```

Note: Name/value pairs are always stored as strings. Remember to convert them to another format when needed!

The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter:

Example

```
if (localStorage.clickcount) {  
    localStorage.clickcount = Number(localStorage.clickcount) + 1;  
} else {  
    localStorage.clickcount = 1;  
}  
document.getElementById("result").innerHTML = "You have clicked the button  
" +  
localStorage.clickcount + " time(s).";
```

The sessionStorage Object

The `sessionStorage` object is equal to the `localStorage` object, **except** that it stores the data for only one session. The data is deleted when the user closes the specific browser tab.

The following example counts the number of times a user has clicked a button, in the current session:

Example

```
if (sessionStorage.clickcount) {  
    sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;  
} else {  
    sessionStorage.clickcount = 1;  
}  
document.getElementById("result").innerHTML = "You have clicked the button  
" +  
sessionStorage.clickcount + " time(s) in this session.";
```

HTML5 Web Workers

A web worker is a JavaScript running in the background, without affecting the performance of the page.

What is a Web Worker?

When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

Browser Support

The numbers in the table specify the first browser version that fully support Web Workers.

API					
Web Workers	4.0	10.0	3.5	4.0	11.5

HTML Web Workers Example

The example below creates a simple web worker that count numbers in the background:

Example

Count numbers:

Check Web Worker Support

Before creating a web worker, check whether the user's browser supports it:

```
if (typeof(Worker) !== "undefined") {      // Yes! Web worker support!
  Some code.... } else {      // Sorry! No Web Worker support.. }
```

Create a Web Worker File

Now, let's create our web worker in an external JavaScript.

Here, we create a script that counts. The script is stored in the "demo_workers.js" file:

```
var i = 0; function timedCount() { i = i + 1; postMessage(i);  
setTimeout("timedCount()",500); } timedCount();
```

The important part of the code above is the `postMessage()` method - which is used to post a message back to the HTML page.

Note: Normally web workers are not used for such simple scripts, but for more CPU intensive tasks.

Create a Web Worker Object

Now that we have the web worker file, we need to call it from an HTML page.

The following lines checks if the worker already exists, if not - it creates a new web worker object and runs the code in "demo_workers.js":

```
if (typeof(w) == "undefined") { w = new Worker("demo_workers.js"); }
```

Then we can send and receive messages from the web worker.

Add an "onmessage" event listener to the web worker.

```
w.onmessage = function(event){ document.getElementById("result").innerHTML  
= event.data; };
```

When the web worker posts a message, the code within the event listener is executed. The data from the web worker is stored in `event.data`.

Terminate a Web Worker

When a web worker object is created, it will continue to listen for messages (even after the external script is finished) until it is terminated.

To terminate a web worker, and free browser/computer resources, use the `terminate()` method:

```
w.terminate();
```

Reuse the Web Worker

If you set the `worker` variable to `undefined`, after it has been terminated, you can reuse the code:

```
w = undefined;
```

Full Web Worker Example Code

We have already seen the Worker code in the .js file. Below is the code for the HTML page:

Example

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

<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>

<script>
var w;

function startWorker() {
    if(typeof(Worker) !== "undefined") {
        if(typeof(w) == "undefined") {
            w = new Worker("demo_workers.js");
        }
        w.onmessage = function(event) {
            document.getElementById("result").innerHTML = event.data;
        };
    } else {
        document.getElementById("result").innerHTML = "Sorry! No Web Worker support.";
    }
}

function stopWorker() {
    w.terminate();
    w = undefined;
}
</script>

</body>
</html>
```

Web Workers and the DOM

Since web workers are in external files, they do not have access to the following JavaScript objects:

- The window object
- The document object
- The parent object

HTML5 Server-Sent Events

Server-Sent Events allow a web page to get updates from a server.

Server-Sent Events - One Way Messaging

A server-sent event is when a web page automatically gets updates from a server. This was also possible before, but the web page would have to ask if any updates were available. With server-sent events, the updates come automatically.
Examples: Facebook/Twitter updates, stock price updates, news feeds, sport results, etc.

Browser Support

The numbers in the table specify the first browser version that fully support server-sent events.

API					
SSE	6.0	Not supported	6.0	5.0	11.5

Receive Server-Sent Event Notifications

The EventSource object is used to receive server-sent event notifications:

Example

```
var source = new EventSource("demo_sse.php");
source.onmessage = function(event) {
    document.getElementById("result").innerHTML += event.data + "<br>";
};
```

Example explained:

- Create a new EventSource object, and specify the URL of the page sending the updates (in this example "demo_sse.php")
- Each time an update is received, the onmessage event occurs
- When an onmessage event occurs, put the received data into the element with id="result"

Check Server-Sent Events Support

In the tryit example above there were some extra lines of code to check browser support for server-sent events:

```
if(typeof(EventSource) !== "undefined") {      // Yes! Server-sent events
    support!      // Some code..... } else {      // Sorry! No server-sent events
    support.. }
```

Server-Side Code Example

For the example above to work, you need a server capable of sending data updates (like PHP or ASP).

The server-side event stream syntax is simple. Set the "Content-Type" header to "text/event-stream". Now you can start sending event streams.

Code in PHP (demo_sse.php):

```
<?phpheader('Content-Type: text/event-stream'); header('Cache-Control: no-
cache');$time = date('r');echo "data: The server time is:
{$time}\n\n";flush();?>
```

Code in ASP (VB) (demo_sse.asp):

```
<%Response.ContentType = "text/event-stream"Response.Expires = -1
Response.Write("data: The server time is: " & now())Response.Flush()%>
```

Code explained:

- Set the "Content-Type" header to "text/event-stream"
- Specify that the page should not cache
- Output the data to send (**Always** start with "data: ")
- Flush the output data back to the web page

The EventSource Object

In the examples above we used the onmessage event to get messages. But other events are also available:

Events	Description
onopen	When a connection to the server is opened
onmessage	When a message is received
onerror	When an error occurs

HTML Graphics

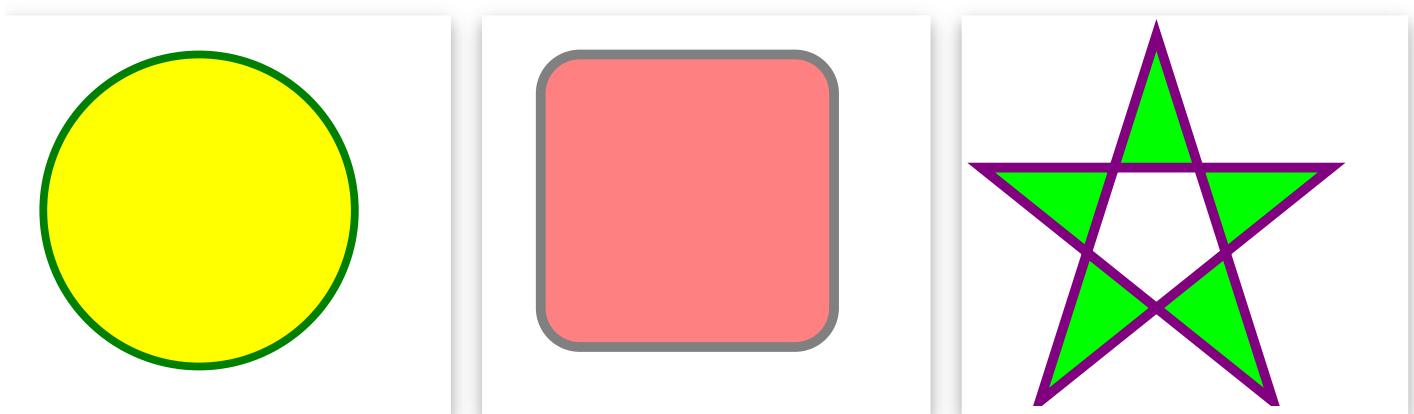
Google Maps

Google Maps lets you embed maps in HTML:



SVG

The HTML <svg> element allows vector based graphics in HTML:



HTML Canvas

The HTML <canvas> element can be used to draw graphics on a web page:

The graphic below is created with <canvas>:



It shows four elements: a red rectangle, a gradient rectangle, a multicolor rectangle, and a multicolor text.

Google Maps Tutorial

Google Maps API

This tutorial is about the Google Maps API (**A**pplication **P**rogramming **I**nterface).

An API is a set of methods and tools that can be used for building software applications.

Google Maps API allows you to display maps on your web site:



Google Maps in HTML

This example creates a Google Map in HTML:

Example

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

<h1>My First Google Map</h1>

<div id="googleMap" style="width:100%;height:400px;"></div>

<script>
function myMap() {
var mapProp= {
    center:new google.maps.LatLng(51.508742,-0.120850),
    zoom:5,
};
var map=new google.maps.Map(document.getElementById("googleMap"),mapProp);
}
</script>

<script src="https://maps.googleapis.com/maps/api/js?
key=YOUR_KEY&callback=myMap"></script>

</body>
</html>
```

Google Maps Basic

Create a Basic Google Map

This example creates a Google Map centered in London, England:

Example

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

<h1>My First Google Map</h1>

<div id="googleMap" style="width:100%;height:400px;"></div>

<script>
function myMap() {
var mapProp= {
    center:new google.maps.LatLng(51.508742,-0.120850),
    zoom:5,
};
var map=new google.maps.Map(document.getElementById("googleMap"),mapProp);
}
</script>

<script src="https://maps.googleapis.com/maps/api/js?
key=YOUR_KEY&callback=myMap"></script>

</body>
</html>
```

The rest of this page describes the example above, step by step.

The Map Container and Size

The map needs an HTML element to hold the map:

```
<div id="googleMap" style="width:100%;height:400px"></div>
```

Also set the size of the map.

Create a Function to Set The Map Properties

```
function myMap() {  
var mapProp= {  
    center:new google.maps.LatLng(51.508742,-0.120850),  
    zoom:5,  
};  
var map=new google.maps.Map(document.getElementById("googleMap"),mapProp);  
}
```

The **mapProp** variable defines the properties for the map.

The **center** property specifies where to center the map (using latitude and longitude coordinates).

The **zoom** property specifies the zoom level for the map (try to experiment with the zoom level).

The line: **var map=new google.maps.Map(document.getElementById("googleMap"), mapProp);** creates a new map inside the <div> element with id="googleMap", using the parameters that are passed (mapProp).

Multiple Maps

The example below defines four maps with different map types:

Example

```
var map1 = new google.maps.Map(document.getElementById("googleMap1"),  
mapOptions1);  
var map2 = new google.maps.Map(document.getElementById("googleMap2"),  
mapOptions2);  
var map3 = new google.maps.Map(document.getElementById("googleMap3"),  
mapOptions3);  
var map4 = new google.maps.Map(document.getElementById("googleMap4"),  
mapOptions4);
```

Free Google API Key

Google allows a website to call any Google API for free, thousands of times a day.

Go to <https://developers.google.com/maps/documentation/javascript/get-api-key> to learn how to get an API key.

Google Maps expects to find the API key in the **key** parameter when loading an API:

```
<script src="https://maps.googleapis.com/maps/api/js?  
key=YOUR_KEY&callback=myMap"></script>
```

Google Maps Overlays

Add a marker to the Google map:



Google Maps - Overlays

Overlays are objects on the map that are bound to latitude/longitude coordinates.

Google Maps has several types of overlays:

- Marker - Single locations on a map. Markers can also display custom icon images
- Polyline - Series of straight lines on a map
- Polygon - Series of straight lines on a map, and the shape is "closed"
- Circle and Rectangle
- Info Windows - Displays content within a popup balloon on top of a map
- Custom overlays

Google Maps - Add a Marker

The Marker constructor creates a marker. (Note that the position property must be set for the marker to display).

Add the marker to the map by using the setMap() method:

Example

```
var marker = new google.maps.Marker({position: myCenter});  
  
marker.setMap(map);
```

Google Maps - Animate the Marker

The example below shows how to animate the marker with the animation property:

Example

```
var marker = new google.maps.Marker({  
  position:myCenter,  
  animation:google.maps.Animation.BOUNCE  
});  
  
marker.setMap(map);
```

Google Maps - Icon Instead of Marker

The example below specifies an image (icon) to use instead of the default marker:

Example

```
var marker=new google.maps.Marker({  
  position:myCenter,  
  icon:'pinkball.png'  
});  
  
marker.setMap(map);
```

Google Maps - Polyline

A Polyline is a line that is drawn through a series of coordinates in an ordered sequence.

A polyline supports the following properties:

- path - specifies several latitude/longitude coordinates for the line
- strokeColor - specifies a hexadecimal color for the line (format: "#FFFFFF")
- strokeOpacity - specifies the opacity of the line (a value between 0.0 and 1.0)
- strokeWeight - specifies the weight of the line's stroke in pixels
- editable - defines whether the line is editable by users (true/false)

Example

```
var myTrip = [stavanger,amsterdam,london];
var flightPath = new google.maps.Polyline({
  path:myTrip,
  strokeColor:"#0000FF",
  strokeOpacity:0.8,
  strokeWeight:2
});
```

Google Maps - Polygon

A Polygon is similar to a Polyline in that it consists of a series of coordinates in an ordered sequence. However, polygons are designed to define regions within a closed loop.

Polygons are made of straight lines, and the shape is "closed" (all the lines connect up).

A polygon supports the following properties:

- path - specifies several LatLng coordinates for the line (first and last coordinate are equal)
- strokeColor - specifies a hexadecimal color for the line (format: "#FFFFFF")
- strokeOpacity - specifies the opacity of the line (a value between 0.0 and 1.0)
- strokeWeight - specifies the weight of the line's stroke in pixels
- fillColor - specifies a hexadecimal color for the area within the enclosed region (format: "#FFFFFF")
- fillOpacity - specifies the opacity of the fill color (a value between 0.0 and 1.0)
- editable - defines whether the line is editable by users (true/false)

Example

```
var myTrip = [stavanger,amsterdam,london,stavanger];
var flightPath = new google.maps.Polygon({
  path:myTrip,
  strokeColor:"#0000FF",
  strokeOpacity:0.8,
  strokeWeight:2,
  fillColor:"#0000FF",
  fillOpacity:0.4
});
```

Google Maps - Circle

A circle supports the following properties:

- center - specifies the google.maps.LatLng of the center of the circle
- radius - specifies the radius of the circle, in meters
- strokeColor - specifies a hexadecimal color for the line around the circle (format: "#FFFFFF")
- strokeOpacity - specifies the opacity of the stroke color (a value between 0.0 and 1.0)
- strokeWeight - specifies the weight of the line's stroke in pixels
- fillColor - specifies a hexadecimal color for the area within the circle (format: "#FFFFFF")
- fillOpacity - specifies the opacity of the fill color (a value between 0.0 and 1.0)
- editable - defines whether the circle is editable by users (true/false)

Example

```
var myCity = new google.maps.Circle({  
  center:amsterdam,  
  radius:20000,  
  strokeColor:"#0000FF",  
  strokeOpacity:0.8,  
  strokeWeight:2,  
  fillColor:"#0000FF",  
  fillOpacity:0.4  
});
```

Google Maps - InfoWindow

Show an InfoWindow with some text content for a marker:

Example

```
var infowindow = new google.maps.InfoWindow({  
  content:"Hello World!"  
});  
  
infowindow.open(map,marker);
```

Google Maps Events

Click the marker to zoom - attach event handlers to Google maps.



Click The Marker to Zoom

We still use the map from the previous page: a map centered on London, England.

Now we want to zoom when a user is clicking on the marker (We attach an event handler to a marker that zooms the map when clicked).

Here is the added code:

Example

```
// Zoom to 9 when clicking on marker
google.maps.event.addListener(marker, 'click', function() {
  map.setZoom(9);
  map.setCenter(marker.getPosition());
});
```

We register for event notifications using the `addListener()` event handler. That method takes an object, an event to listen for, and a function to call when the specified event occurs.

Pan Back to Marker

Here, we save the zoom changes and pan the map back after 3 seconds:

Example

```
google.maps.event.addListener(marker, 'click', function() {
  var pos = map.getZoom();
  map.setZoom(9);
  map.setCenter(marker.getPosition());
  window.setTimeout(function() {map.setZoom(pos);}, 3000);
});
```

Open an InfoWindow When Clicking on The Marker

Click on the marker to show an infowindow with some text:

Example

```
var infowindow = new google.maps.InfoWindow({
  content:"Hello World!"
});

google.maps.event.addListener(marker, 'click', function() {
  infowindow.open(map,marker);
});
```

Set Markers and Open InfoWindow for Each Marker

Run a function when the user clicks on the map.

The placeMarker() function places a marker where the user has clicked, and shows an infowindow with the latitudes and longitudes of the marker:

Example

```
google.maps.event.addListener(map, 'click', function(event) {
  placeMarker(map, event.latLng);
});

function placeMarker(map, location) {
  var marker = new google.maps.Marker({
    position: location,
    map: map
 });
  var infowindow = new google.maps.InfoWindow({
    content: 'Latitude: ' + location.lat() +
    '<br>Longitude: ' + location.lng()
 });
  infowindow.open(map,marker);
}
```

Google Maps Controls

A Google map - with the default control set:



Google Maps - The Default Controls

When showing a standard Google map, it comes with the default control set:

- Zoom - displays a slider or "+/-" buttons to control the zoom level of the map
- Pan - displays a pan control for panning the map
- MapType - lets the user toggle between map types (roadmap and satellite)
- Street View - displays a Pegman icon which can be dragged to the map to enable Street View

Google Maps - More Controls

In addition to the default controls, Google Maps also has:

- Scale - displays a map scale element
- Rotate - displays a small circular icon which allows you to rotate maps
- Overview Map - displays a thumbnail overview map reflecting the current map viewport within a wider area

You can specify which controls to show when creating the map (inside `MapOptions`) or by calling `setOptions()` to change the map's options.

Google Maps - Disabling The Default Controls

You may instead wish to turn off the default controls.

To do so, set the Map's disableDefaultUI property (within the Map options object) to true:

Example

```
var mapOptions {disableDefaultUI: true}
```

Google Maps - Turn On All Controls

Some controls appear on the map by default; while others will not appear unless you set them.

Adding or removing controls from the map is specified in the Map options object.

Set the control to true to make it visible - Set the control to false to hide it.

The following example turns "on" all controls:

Example

```
var mapOptions {  
    panControl: true,  
    zoomControl: true,  
    mapTypeControl: true,  
    scaleControl: true,  
    streetViewControl: true,  
    overviewMapControl: true,  
    rotateControl: true  
}
```

Google Maps - Modifying Controls

Several of the map controls are configurable.

The controls can be modified by specifying control options fields.

For example, options for modifying a Zoom control are specified in the zoomControlOptions field. The zoomControlOptions field may contain:

- google.maps.ZoomControlStyle.SMALL - displays a mini-zoom control (only + and - buttons)
- google.maps.ZoomControlStyle.LARGE - displays the standard zoom slider control
- google.maps.ZoomControlStyle.DEFAULT - picks the best zoom control based on device and map size

Example

```
zoomControl: true,  
zoomControlOptions: {  
    style: google.maps.ZoomControlStyle.SMALL  
}
```

Note: If you modify a control, always enable the control first (set it to true).

Another configurable control is the MapType control.

Options for modifying a control are specified in the mapTypeControlOptions field. The mapTypeControlOptions field may contain::

- google.maps.MapTypeControlStyle.HORIZONTAL_BAR - display one button for each map type
- google.maps.MapTypeControlStyle.Dropdown_MENU - select map type via a dropdown menu
- google.maps.MapTypeControlStyle.DEFAULT - displays the "default" behavior (depends on screen size)

Example

```
mapTypeControl: true,  
mapTypeControlOptions: {  
    style: google.maps.MapTypeControlStyle.Dropdown_MENU  
}
```

You can also position a control, with the ControlPosition property:

Example

```
mapTypeControl: true,  
mapTypeControlOptions: {  
    style: google.maps.MapTypeControlStyle.DROPDOWN_MENU,  
    position: google.maps.ControlPosition.TOP_CENTER  
}
```

Google Maps Types

A Google map of type HYBRID:



Google Maps - Basic Map Types

The following map types are supported in Google Maps API:

- ROADMAP (normal, default 2D map)
- SATELLITE (photographic map)
- HYBRID (photographic map + roads and city names)
- TERRAIN (map with mountains, rivers, etc.)

The map type is specified either within the Map properties object, with the mapTypeId property:

```
var mapOptions = {
  center: new google.maps.LatLng(51.508742, -0.120850),
  zoom: 7,
  mapTypeId: google.maps.MapTypeId.HYBRID
};
```

Or by calling the map's setMapTypeId() method:

```
map.setMapTypeId(google.maps.MapTypeId.HYBRID);
```

Google Maps - 45° Perspective View

The map types SATELLITE and HYBRID support a 45° perspective imagery view for certain locations (only at high zoom levels).

If you zoom into a location with 45° imagery view, the map will automatically alter the perspective view. In addition, the map will add:

- A compass wheel around the Pan control, allowing you to rotate the image
- A Rotate control between the Pan and Zoom controls, allowing you to rotate the image 90°
- A toggle control for displaying the 45° perspective view, under the Satellite control/label

Note: Zooming out from a map with 45° imagery will revert each of these changes, and the original map is displayed.

The following example shows a 45° perspective view of Palazzo Ducale in Venice, Italy:

Example

```
var mapOptions = {  
  center:myCenter,  
  zoom:18,  
  mapTypeId:google.maps.MapTypeId.HYBRID  
};
```

Google Maps - Disable 45° Perspective View - setTilt(0)

You can disable 45° perspective view by calling `setTilt(0)` on the Map object:

Example

```
map.setTilt(0);
```

Tip: To enable 45° perspective view at a later point, call `setTilt(45)`.

Google Maps Reference

The Map() Constructor

Example

Create a Google Map:

```
var map = new google.maps.Map(mapCanvas, mapOptions);
```

Definition and Usage

The Map() constructor creates a new map inside a specified HTML element (typically a <div> element).

Syntax

```
new google.maps.Map(HTMLElement,MapOptions)
```

Parameter Values

Parameter	Description
<i>HTMLElement</i>	Specifies in what HTML element to put the map
<i>MapOptions</i>	A MapOptions object that holds the map initialization variables/options

Methods of Map()

Method	Return Value	Description
<code>fitBounds(<i>LatLangBounds</i>)</code>	None	Sets the viewport to contain the given bounds
<code>getBounds()</code>	<i>LatLang</i> , <i>LatLang</i>	Returns the south-west latitude/longitude and the north-east latitude/longitude of the current viewport

<code>getCenter()</code>	<i>LatLang</i>	Returns the lat/lng of the center of the map
<code>getDiv()</code>	<i>Node</i>	Returns a DOM object that contains the map
<code>getHeading()</code>	<i>number</i>	Returns the compass heading of aerial imagery (for SATELLITE and HYBRID map types)
<code>getMapTypeId()</code>	HYBRID ROADMAP SATELLITE TERRAIN	Returns the current map type
<code>getProjection()</code>	<i>Projection</i>	Returns the current Projection
<code>getStreetView()</code>	<i>StreetViewPanorama</i>	Returns the default StreetViewPanorama bound to the map
<code>getTilt()</code>	<i>number</i>	Returns the angle of incidence for aerial imagery in degrees (for SATELLITE and HYBRID map types)
<code>getZoom()</code>	<i>number</i>	Returns the current zoom level of the map
<code>panBy(<i>xnumber,ynumber</i>)</code>	None	Changes the center of the map by the given distance in pixels
<code>panTo(<i>LatLang</i>)</code>	None	Changes the center of the map to the given LatLang
<code>panToBounds(<i>LatLangBounds</i>)</code>	None	Pans the map by the minimum amount necessary to contain the given LatLangBounds
<code>setCenter(<i>LatLang</i>)</code>	None	Sets the lat/lng of the center of the map
<code>setHeading(<i>number</i>)</code>	None	Sets the compass heading for aerial imagery measured in degrees from cardinal direction North
<code>setMapTypeId(<i>MapTypeId</i>)</code>	None	Sets the map type to display
<code>setOptions(<i>MapOptions</i>)</code>	None	

<code>setStreetView(StreetViewPanorama)</code>	None	Binds a StreetViewPanorama to the map
<code>setTilt(number)</code>	None	Sets the angle of incidence for aerial imagery in degrees (for SATELLITE and HYBRID map types)
<code>setZoom(number)</code>	None	Sets the zoom level of the map

Properties of Map()

Property	Type	Description
<code>controls</code>	<code>Array.<MVCArray.<Node>></code>	Additional controls to attach to the map
<code>mapTypes</code>	<code>MapTypeRegistry</code>	A registry of MapType instances by string ID
<code>overlayMapTypes</code>	<code>MVCArray.<MapType></code>	Additional map types to overlay

Events of Map()

Event	Arguments	Description
bounds_changed	None	Fired when the viewport bounds have changed
center_changed	None	Fired when the map center property changes
click	<i>MouseEvent</i>	Fired when the user clicks on the map
dblclick	<i>MouseEvent</i>	Fired when the user double-clicks on the map
drag	None	Fired repeatedly while the user drags the map
dragend	None	Fired when the user stops dragging the map
dragstart	None	Fired when the user starts dragging the map
heading_changed	None	Fired when the map heading property changes
idle	None	Fired when the map becomes idle after panning or zooming
maptyp eid _changed	None	Fired when the mapTypeId property changes
mousemove	<i>MouseEvent</i>	Fired whenever the user's mouse moves over the map container
mouseout	<i>MouseEvent</i>	Fired when the user's mouse exits the map container
mouseover	<i>MouseEvent</i>	Fired when the user's mouse enters the map container
projection_changed	None	Fired when the projection has changed
resize	None	Fired when the map (div) changes size
rightclick	<i>MouseEvent</i>	Fired when the user right-clicks on the map
tilesloaded	None	Fired when the visible tiles have finished loading
tilt_changed	None	Fired when the map tilt property changes
zoom_changed	None	Fired when the map zoom property changes

Overlays

Constructor/Object	Description
Marker	Creates a marker. (Note that the position must be set for the marker to display)
MarkerOptions	Options for rendering the marker
MarkerImage	A structure representing a Marker icon or shadow image
MarkerShape	Defines the marker shape to use in determination of a marker's clickable region (type and coord)
Animation	Specifies animations that can be played on a marker (bounce or drop)
InfoWindow	Creates an info window
InfoWindowOptions	Options for rendering the info window
Polyline	Creates a polyline (contains path and stroke styles)
PolylineOptions	Options for rendering the polyline
Polygon	Creates a polygon (contains path and stroke+fill styles)
PolygonOptions	Options for rendering the polygon
Rectangle	Creates a rectangle (contains bounds and stroke+fill styles)
RectangleOptions	Options for rendering the rectangle
Circle	Creates a circle (contains center+radius and stroke+fill styles)
CircleOptions	Options for rendering the circle
GroundOverlay	
GroundOverlayOptions	
OverlayView	
MapPanes	
MapCanvasProjection	

Events

Constructor/Object	Description
MapsEventListener	It has no methods and no constructor. Its instances are returned from addListener(), addDomListener() and are eventually passed back to removeListener()
event	Adds/Removes/Trigger event listeners
MouseEvent	Returned from various mouse events on the map and overlays

Controls

Constructor/Object	Description
MapTypeControlOptions	Holds options for modifying a control (position and style)
MapTypeControlStyle	Specifies what kind of map control to display (Drop-down menu or buttons)
OverviewMapControlOptions	Options for rendering of the overview map control (opened or collapsed)
PanControlOptions	Options for rendering of the pan control (position)
RotateControlOptions	Options for rendering of the rotate control (position)
ScaleControlOptions	Options for rendering of the scale control (position and style)
ScaleControlStyle	Specifies what kind of scale control to display
StreetViewControlOptions	Options for rendering of the street view pegman control (position)
ZoomControlOptions	Options for rendering of the zoom control (position and style)
ZoomControlStyle	Specifies what kind of zoom control to display (large or small)
ControlPosition	Specifies the placement of controls on the map

SVG Tutorial

SVG stands for Scalable Vector Graphics.

SVG defines vector-based graphics in XML format.

Examples in Each Chapter

With our "Try it Yourself" editor, you can edit the SVG, and click on a button to view the result.

SVG Example

```
<html>
<body>

<h1>My first SVG</h1>

<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4"
fill="yellow" />
</svg>

</body>
</html>
```

What you should already know

Before you continue, you should have some basic understanding of the following:

- HTML
- Basic XML

If you want to study these subjects first, find the tutorials on our [Home page](#).

What is SVG?

- SVG stands for Scalable Vector Graphics
- SVG is used to define vector-based graphics for the Web
- SVG defines the graphics in XML format
- Every element and every attribute in SVG files can be animated
- SVG is a W3C recommendation
- SVG integrates with other W3C standards such as the DOM and XSL

SVG is a W3C Recommendation

SVG 1.0 became a W3C Recommendation on 4 September 2001.

SVG 1.1 became a W3C Recommendation on 14 January 2003.

SVG 1.1 (Second Edition) became a W3C Recommendation on 16 August 2011.

SVG Advantages

Advantages of using SVG over other image formats (like JPEG and GIF) are:

- SVG images can be created and edited with any text editor
- SVG images can be searched, indexed, scripted, and compressed
- SVG images are scalable
- SVG images can be printed with high quality at any resolution
- SVG images are zoomable
- SVG graphics do NOT lose any quality if they are zoomed or resized
- SVG is an open standard
- SVG files are pure XML

The main competitor to SVG is Flash.

The biggest advantage SVG has over Flash is the compliance with other standards (e.g. XSL and the DOM). Flash relies on proprietary technology that is not open source.

Creating SVG Images

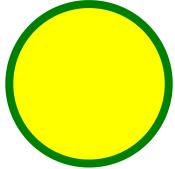
SVG images can be created with any text editor, but it is often more convenient to create SVG images with a drawing program, like [Inkscape](#).

SVG in HTML

In HTML5, you can embed SVG elements directly into your HTML pages.

Embed SVG Directly Into HTML Pages

Here is an example of a simple SVG graphic:



and here is the HTML code:

Example

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

<h1>My first SVG</h1>

<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4"
fill="yellow" />
</svg>

</body>
</html>
```

SVG Code explanation:

- An SVG image begins with an `<svg>` element
- The width and height attributes of the `<svg>` element define the width and height of the SVG image
- The `<circle>` element is used to draw a circle
- The `cx` and `cy` attributes define the x and y coordinates of the center of the circle. If `cx` and `cy` are not set, the circle's center is set to (0, 0)
- The `r` attribute defines the radius of the circle
- The `stroke` and `stroke-width` attributes control how the outline of a shape appears. We set the outline of the circle to a 4px green "border"
- The `fill` attribute refers to the color inside the circle. We set the fill color to yellow
- The closing `</svg>` tag closes the SVG image

SVG <rect>

SVG Shapes

SVG has some predefined shape elements that can be used by developers:

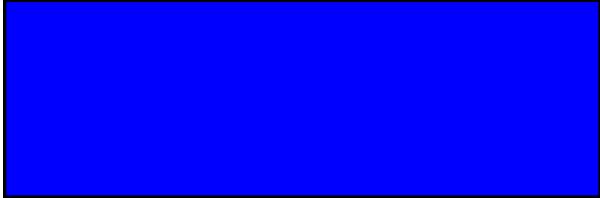
- Rectangle <rect>
- Circle <circle>
- Ellipse <ellipse>
- Line <line>
- Polyline <polyline>
- Polygon <polygon>
- Path <path>

The following chapters will explain each element, starting with the rect element.

SVG Rectangle - <rect>

Example 1

The <rect> element is used to create a rectangle and variations of a rectangle shape:



Here is the SVG code:

Example

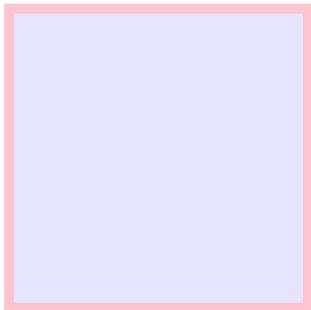
```
<svg width="400" height="110">
  <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-
width:3;stroke:rgb(0,0,0)" />
</svg>
```

Code explanation:

- The width and height attributes of the <rect> element define the height and the width of the rectangle
- The style attribute is used to define CSS properties for the rectangle
- The CSS fill property defines the fill color of the rectangle
- The CSS stroke-width property defines the width of the border of the rectangle
- The CSS stroke property defines the color of the border of the rectangle

Example 2

Let's look at another example that contains some new attributes:



Here is the SVG code:

Example

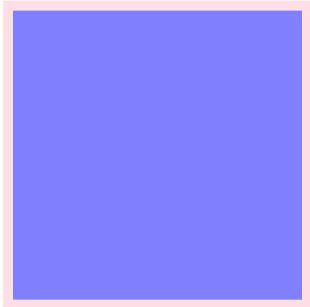
```
<svg width="400" height="180">
  <rect x="50" y="20" width="150" height="150"
    style="fill:blue;stroke:pink;stroke-width:5;fill-opacity:0.1;stroke-
    opacity:0.9" />
</svg>
```

Code explanation:

- The x attribute defines the left position of the rectangle (e.g. x="50" places the rectangle 50 px from the left margin)
- The y attribute defines the top position of the rectangle (e.g. y="20" places the rectangle 20 px from the top margin)
- The CSS fill-opacity property defines the opacity of the fill color (legal range: 0 to 1)
- The CSS stroke-opacity property defines the opacity of the stroke color (legal range: 0 to 1)

Example 3

Define the opacity for the whole element:



Here is the SVG code:

Example

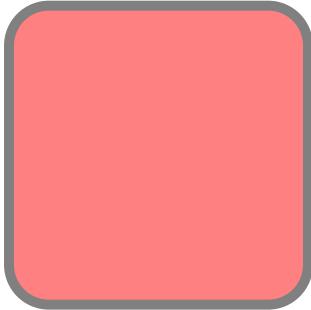
```
<svg width="400" height="180">
  <rect x="50" y="20" width="150" height="150"
    style="fill:blue;stroke:pink;stroke-width:5;opacity:0.5" />
</svg>
```

Code explanation:

- The CSS opacity property defines the opacity value for the whole element (legal range: 0 to 1)

Example 4

Last example, create a rectangle with rounded corners:



Here is the SVG code:

Example

```
<svg width="400" height="180">
  <rect x="50" y="20" rx="20" ry="20" width="150" height="150"
    style="fill:red;stroke:black;stroke-width:5;opacity:0.5" />
</svg>
```

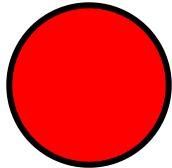
Code explanation:

- The rx and the ry attributes rounds the corners of the rectangle

SVG <circle>

SVG Circle - <circle>

The <circle> element is used to create a circle:



Here is the SVG code:

Example

```
<svg height="100" width="100">
  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red">
  />
</svg>
```

Code explanation:

- The cx and cy attributes define the x and y coordinates of the center of the circle. If cx and cy are omitted, the circle's center is set to (0,0)
- The r attribute defines the radius of the circle

SVG <ellipse>

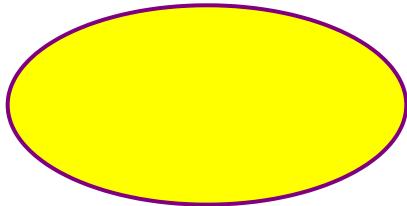
SVG Ellipse - <ellipse>

The <ellipse> element is used to create an ellipse.

An ellipse is closely related to a circle. The difference is that an ellipse has an x and a y radius that differs from each other, while a circle has equal x and y radius:

Example 1

The following example creates an ellipse:



Here is the SVG code:

Example

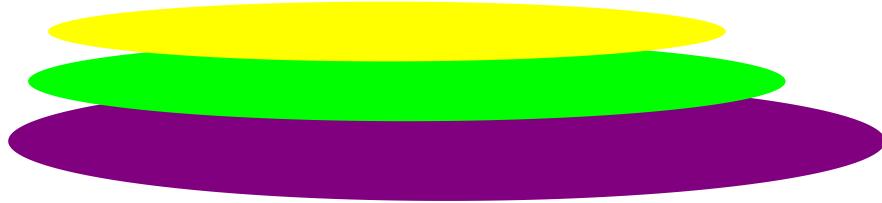
```
<svg height="140" width="500">
  <ellipse cx="200" cy="80" rx="100" ry="50"
    style="fill:yellow;stroke:purple;stroke-width:2" />
</svg>
```

Code explanation:

- The cx attribute defines the x coordinate of the center of the ellipse
- The cy attribute defines the y coordinate of the center of the ellipse
- The rx attribute defines the horizontal radius
- The ry attribute defines the vertical radius

Example 2

The following example creates three ellipses on top of each other:



Here is the SVG code:

Example

```
<svg height="150" width="500">
  <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:purple" />
  <ellipse cx="220" cy="70" rx="190" ry="20" style="fill:lime" />
  <ellipse cx="210" cy="45" rx="170" ry="15" style="fill:yellow" />
</svg>
```

Example 3

The following example combines two ellipses (one yellow and one white):



Here is the SVG code:

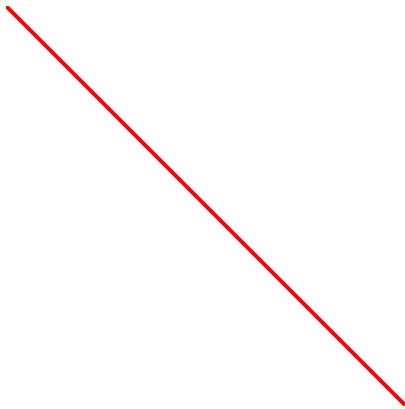
Example

```
<svg height="100" width="500">
  <ellipse cx="240" cy="50" rx="220" ry="30" style="fill:yellow" />
  <ellipse cx="220" cy="50" rx="190" ry="20" style="fill:white" />
</svg>
```

SVG <line>

SVG Line - <line>

The <line> element is used to create a line:



Here is the SVG code:

Example

```
<svg height="210" width="500">
  <line x1="0" y1="0" x2="200" y2="200" style="stroke:rgb(255,0,0);stroke-width:2" />
</svg>
```

Code explanation:

- The x1 attribute defines the start of the line on the x-axis
- The y1 attribute defines the start of the line on the y-axis
- The x2 attribute defines the end of the line on the x-axis
- The y2 attribute defines the end of the line on the y-axis

SVG <polygon>

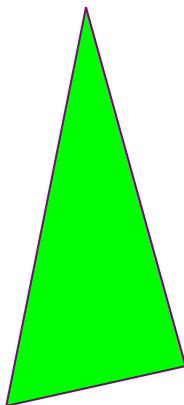
SVG Polygon - <polygon>

The <polygon> element is used to create a graphic that contains at least three sides. Polygons are made of straight lines, and the shape is "closed" (all the lines connect up).

Polygon comes from Greek. "Poly" means "many" and "gon" means "angle".

Example 1

The following example creates a polygon with three sides:



Here is the SVG code:

Example

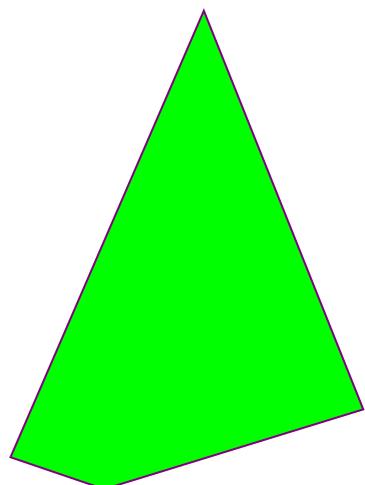
```
<svg height="210" width="500">
  <polygon points="200,10 250,190 160,210"
style="fill:lime;stroke:purple;stroke-width:1" />
</svg>
```

Code explanation:

- The points attribute defines the x and y coordinates for each corner of the polygon

Example 2

The following example creates a polygon with four sides:



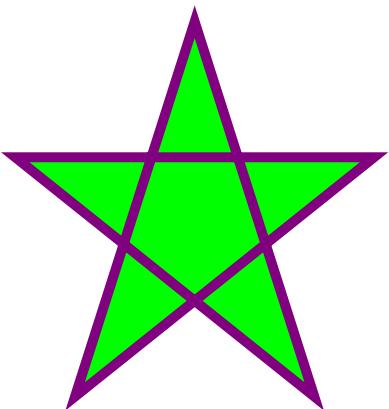
Here is the SVG code:

Example

```
<svg height="250" width="500">
  <polygon points="220,10 300,210 170,250 123,234"
    style="fill:lime;stroke:purple;stroke-width:1" />
</svg>
```

Example 3

Use the <polygon> element to create a star:



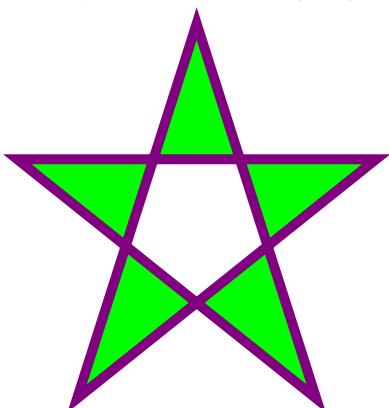
Here is the SVG code:

Example

```
<svg height="210" width="500">
  <polygon points="100,10 40,198 190,78 10,78 160,198"
    style="fill:lime;stroke:purple;stroke-width:5;fill-rule:nonzero;" />
</svg>
```

Example 4

Change the fill-rule property to "evenodd":



Here is the SVG code:

Example

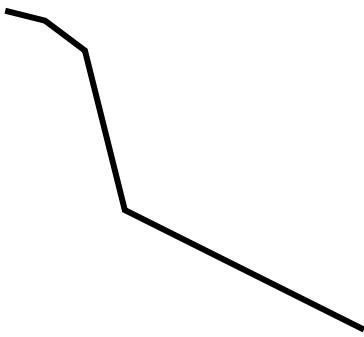
```
<svg height="210" width="500">
  <polygon points="100,10 40,198 190,78 10,78 160,198"
    style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />
</svg>
```

SVG <polyline>

SVG Polyline - <polyline>

Example 1

The <polyline> element is used to create any shape that consists of only straight lines (that is connected at several points):



Here is the SVG code:

Example

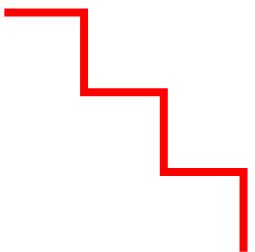
```
<svg height="200" width="500">
  <polyline points="20,20 40,25 60,40 80,120 120,140 200,180"
    style="fill:none;stroke:black;stroke-width:3" />
</svg>
```

Code explanation:

- The points attribute defines the list of points (pairs of x and y coordinates) required to draw the polyline

Example 2

Another example with only straight lines:



Here is the SVG code:

Example

```
<svg height="180" width="500">
  <polyline points="0,40 40,40 40,80 80,80 80,120 120,120 120,160"
    style="fill:white;stroke:red;stroke-width:4" />
</svg>
```

SVG <path>

SVG Path - <path>

The <path> element is used to define a path.

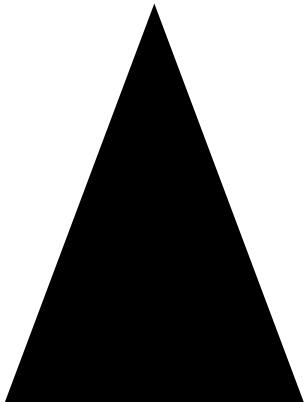
The following commands are available for path data:

- M = moveto
- L = lineto
- H = horizontal lineto
- V = vertical lineto
- C = curveto
- S = smooth curveto
- Q = quadratic Bézier curve
- T = smooth quadratic Bézier curveto
- A = elliptical Arc
- Z = closepath

Note: All of the commands above can also be expressed with lower letters. Capital letters means absolutely positioned, lower cases means relatively positioned.

Example 1

The example below defines a path that starts at position 150,0 with a line to position 75,200 then from there, a line to 225,200 and finally closing the path back to 150,0:



Here is the SVG code:

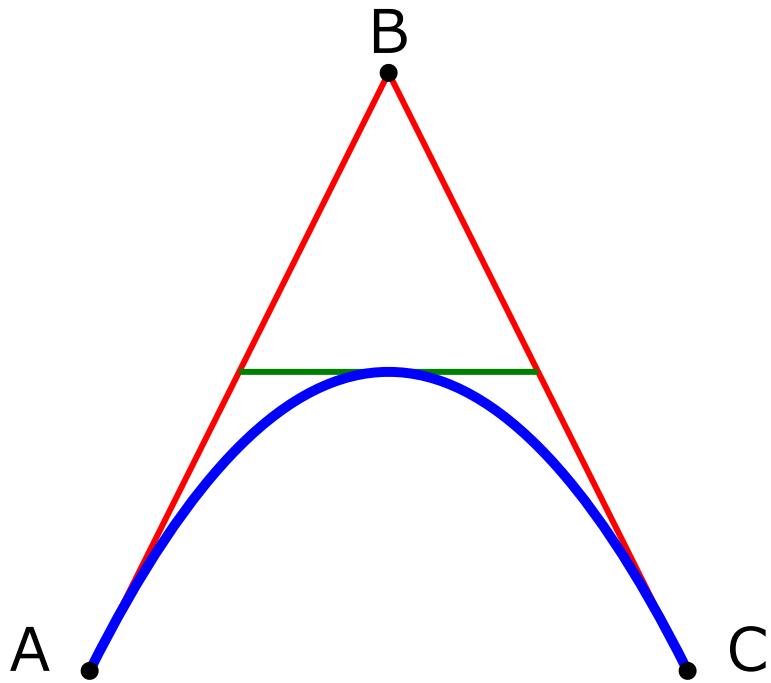
Example

```
<svg height="210" width="400">
  <path d="M150 0 L75 200 L225 200 Z" />
</svg>
```

Example 2

Bézier curves are used to model smooth curves that can be scaled indefinitely. Generally, the user selects two endpoints and one or two control points. A Bézier curve with one control point is called a quadratic Bézier curve and the kind with two control points is called cubic.

The following example creates a quadratic Bézier curve, where A and C are the start and end points, B is the control point:



Here is the SVG code:

Example

```
<svg height="400" width="450">
  <path id="lineAB" d="M 100 350 l 150 -300" stroke="red"
    stroke-width="3" fill="none" />
  <path id="lineBC" d="M 250 50 l 150 300" stroke="red"
    stroke-width="3" fill="none" />
  <path d="M 175 200 l 150 0" stroke="green" stroke-width="3"
    fill="none" />
  <path d="M 100 350 q 150 -300 300 0" stroke="blue"
    stroke-width="5" fill="none" />
  <!-- Mark relevant points -->
  <g stroke="black" stroke-width="3" fill="black">
    <circle id="pointA" cx="100" cy="350" r="3" />
    <circle id="pointB" cx="250" cy="50" r="3" />
    <circle id="pointC" cx="400" cy="350" r="3" />
  </g>
  <!-- Label the points -->
  <g font-size="30" font-family="sans-serif" fill="black" stroke="none"
    text-anchor="middle">
    <text x="100" y="350" dx="-30">A</text>
    <text x="250" y="50" dy="-10">B</text>
    <text x="400" y="350" dx="30">C</text>
  </g>
</svg>
```

Complex? YES!!!! Because of the complexity involved in drawing paths it is highly recommended to use an SVG editor to create complex graphics.

SVG <text>

SVG Text - <text>

The <text> element is used to define a text.

Example 1

Write a text:

I love SVG!

Here is the SVG code:

Example

```
<svg height="30" width="200">
  <text x="0" y="15" fill="red">I love SVG!</text>
</svg>
```

Example 2

Rotate the text:

I love SVG

Here is the SVG code:

Example

```
<svg height="60" width="200">
  <text x="0" y="15" fill="red" transform="rotate(30 20,40)">I love
  SVG</text>
</svg>
```

Example 3

The `<text>` element can be arranged in any number of sub-groups with the `<tspan>` element. Each `<tspan>` element can contain different formatting and position.

Text on several lines (with the `<tspan>` element):

Several lines:

First line.

Second line.

Here is the SVG code:

Example

```
<svg height="90" width="200">
  <text x="10" y="20" style="fill:red;">Several lines:
    <tspan x="10" y="45">First line.</tspan>
    <tspan x="10" y="70">Second line.</tspan>
  </text>
</svg>
```

Example 4

Text as a link (with the `<a>` element):

I love SVG!

Here is the SVG code:

Example

```
<svg height="30" width="200" xmlns:xlink="http://www.w3.org/1999/xlink">
  <a xlink:href="https://www.w3schools.com/graphics/" target="_blank">
    <text x="0" y="15" fill="red">I love SVG!</text>
  </a>
</svg>
```

SVG Stroke Properties

SVG Stroke Properties

SVG offers a wide range of stroke properties. In this chapter we will look at the following:

- stroke
- stroke-width
- stroke-linecap
- stroke-dasharray

All the stroke properties can be applied to any kind of lines, text and outlines of elements like a circle.

SVG stroke Property

The stroke property defines the color of a line, text or outline of an element:



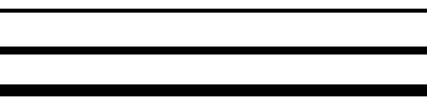
Here is the SVG code:

Example

```
<svg height="80" width="300">
  <g fill="none">
    <path stroke="red" d="M5 20 1215 0" />
    <path stroke="black" d="M5 40 1215 0" />
    <path stroke="blue" d="M5 60 1215 0" />
  </g>
</svg>
```

SVG stroke-width Property

The stroke-width property defines the thickness of a line, text or outline of an element:



Here is the SVG code:

Example

```
<svg height="80" width="300">
  <g fill="none" stroke="black">
    <path stroke-width="2" d="M5 20 1215 0" />
    <path stroke-width="4" d="M5 40 1215 0" />
    <path stroke-width="6" d="M5 60 1215 0" />
  </g>
</svg>
```

SVG stroke-linecap Property

The stroke-linecap property defines different types of endings to an open path:



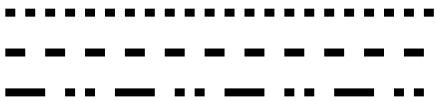
Here is the SVG code:

Example

```
<svg height="80" width="300">
  <g fill="none" stroke="black" stroke-width="6">
    <path stroke-linecap="butt" d="M5 20 1215 0" />
    <path stroke-linecap="round" d="M5 40 1215 0" />
    <path stroke-linecap="square" d="M5 60 1215 0" />
  </g>
</svg>
```

SVG stroke-dasharray Property

The stroke-dasharray property is used to create dashed lines:



Here is the SVG code:

Example

```
<svg height="80" width="300">
  <g fill="none" stroke="black" stroke-width="4">
    <path stroke-dasharray="5,5" d="M5 20 1215 0" />
    <path stroke-dasharray="10,10" d="M5 40 1215 0" />
    <path stroke-dasharray="20,10,5,5,5,10" d="M5 60 1215 0" />
  </g>
</svg>
```

SVG Filters

SVG Filters

SVG Filters are used to add special effects to SVG graphics.

Browser Support

The numbers in the table specify the first browser version that supports SVG filters.

					
SVG Filters	8.0	10.0	3.0	6.0	9.6

SVG Filter Elements

In the next chapters, we will only demonstrate a touch of the filter effects that are possible - and give you an idea of what can be done with SVG.

The available filter elements in SVG are:

- <feBlend> - filter for combining images
- <feColorMatrix> - filter for color transforms
- <feComponentTransfer>
- <feComposite>
- <feConvolveMatrix>
- <feDiffuseLighting>
- <feDisplacementMap>
- <feFlood>
- <feGaussianBlur>
- <feImage>
- <feMerge>
- <feMorphology>
- <feOffset> - filter for drop shadows
- <feSpecularLighting>
- <feTile>
- <feTurbulence>
- <feDistantLight> - filter for lighting
- <fePointLight> - filter for lighting
- <feSpotLight> - filter for lighting

Tip: You can use multiple filters on each SVG element!

SVG Blur Effects

<defs> and <filter>

All internet SVG filters are defined within a <defs> element. The <defs> element is short for definitions and contains definition of special elements (such as filters).

The <filter> element is used to define an SVG filter. The <filter> element has a required id attribute which identifies the filter. The graphic then points to the filter to use.

SVG <feGaussianBlur>

Example 1

The <feGaussianBlur> element is used to create blur effects:



Here is the SVG code:

Example

```
<svg height="110" width="110">
  <defs>
    <filter id="f1" x="0" y="0">
      <feGaussianBlur in="SourceGraphic" stdDeviation="15" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
    fill="yellow" filter="url(#f1)" />
</svg>
```

Code explanation:

- The id attribute of the <filter> element defines a unique name for the filter
- The blur effect is defined with the <feGaussianBlur> element
- The in="SourceGraphic" part defines that the effect is created for the entire element
- The stdDeviation attribute defines the amount of the blur
- The filter attribute of the <rect> element links the element to the "f1" filter

SVG Drop Shadows

<defs> and <filter>

All SVG filters are defined within a <defs> element. The <defs> element is short for definitions and contains definition of special elements (such as filters).

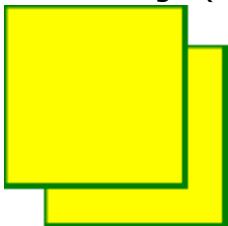
The <filter> element is used to define an SVG filter. The <filter> element has a required id attribute which identifies the filter. The graphic then points to the filter to use.

SVG <feOffset>

Example 1

The <feOffset> element is used to create drop shadow effects. The idea is to take an SVG graphic (image or element) and move it a little bit in the xy plane.

The first example offsets a rectangle (with <feOffset>), then blend the original on top of the offset image (with <feBlend>):



Here is the SVG code:

Example

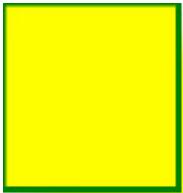
```
<svg height="120" width="120">
  <defs>
    <filter id="f1" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceGraphic" dx="20" dy="20" />
      <feBlend in="SourceGraphic" in2="offOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
        fill="yellow" filter="url(#f1)" />
</svg>
```

Code explanation:

- The id attribute of the <filter> element defines a unique name for the filter
- The filter attribute of the <rect> element links the element to the "f1" filter

Example 2

Now, the offset image can be blurred (with `<feGaussianBlur>`):



Here is the SVG code:

Example

```
<svg height="140" width="140">
  <defs>
    <filter id="f2" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceGraphic" dx="20" dy="20" />
      <feGaussianBlur result="blurOut" in="offOut" stdDeviation="10" />
      <feBlend in="SourceGraphic" in2="blurOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
        fill="yellow" filter="url(#f2)" />
</svg>
```

Code explanation:

- The `stdDeviation` attribute of the `<feGaussianBlur>` element defines the amount of the blur

Example 3

Now, make the shadow black:



Here is the SVG code:

Example

```
<svg height="140" width="140">
  <defs>
    <filter id="f3" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceAlpha" dx="20" dy="20" />
      <feGaussianBlur result="blurOut" in="offOut" stdDeviation="10" />
      <feBlend in="SourceGraphic" in2="blurOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
        fill="yellow" filter="url(#f3)" />
</svg>
```

Code explanation:

- The in attribute of the `<feOffset>` element is changed to "SourceAlpha" which uses the Alpha channel for the blur instead of the entire RGBA pixel

Example 4

Now, treat the shadow as a color:



Here is the SVG code:

Example

```
<svg height="140" width="140">
  <defs>
    <filter id="f4" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceGraphic" dx="20" dy="20" />
      <feColorMatrix result="matrixOut" in="offOut" type="matrix"
        values="0.2 0 0 0 0 0 0.2 0 0 0 0 0 0.2 0 0 0 0 0 1 0" />
      <feGaussianBlur result="blurOut" in="matrixOut" stdDeviation="10" />
      <feBlend in="SourceGraphic" in2="blurOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
    fill="yellow" filter="url(#f4)" />
</svg>
```

Code explanation:

- The `<feColorMatrix>` filter is used to transform the colors in the offset image closer to black. The three values of '0.2' in the matrix all get multiplied by the red, green and blue channels. Reducing their values brings the colors closer to black (black is 0)

SVG Gradients - Linear

SVG Gradients

A gradient is a smooth transition from one color to another. In addition, several color transitions can be applied to the same element.

There are two main types of gradients in SVG:

- Linear
- Radial

SVG Linear Gradient - <linearGradient>

The <linearGradient> element is used to define a linear gradient.

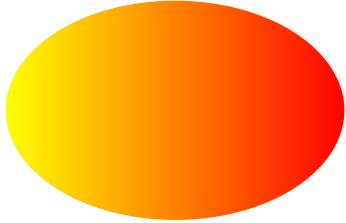
The <linearGradient> element must be nested within a <defs> tag. The <defs> tag is short for definitions and contains definition of special elements (such as gradients).

Linear gradients can be defined as horizontal, vertical or angular gradients:

- Horizontal gradients are created when y1 and y2 are equal and x1 and x2 differ
- Vertical gradients are created when x1 and x2 are equal and y1 and y2 differ
- Angular gradients are created when x1 and x2 differ and y1 and y2 differ

Example 1

Define an ellipse with a horizontal linear gradient from yellow to red:



Here is the SVG code:

Example

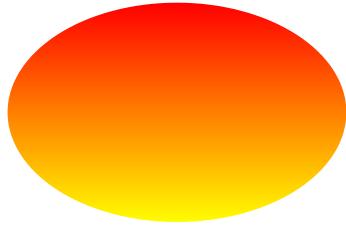
```
<svg height="150" width="400">
  <defs>
    <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

Code explanation:

- The id attribute of the `<linearGradient>` tag defines a unique name for the gradient
- The x1, x2, y1,y2 attributes of the `<linearGradient>` tag define the start and end position of the gradient
- The color range for a gradient can be composed of two or more colors. Each color is specified with a `<stop>` tag. The offset attribute is used to define where the gradient color begin and end
- The fill attribute links the ellipse element to the gradient

Example 2

Define an ellipse with a vertical linear gradient from yellow to red:



Here is the SVG code:

Example

```
<svg height="150" width="400">
  <defs>
    <linearGradient id="grad2" x1="0%" y1="0%" x2="0%" y2="100%">
      <stop offset="0%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad2)" />
</svg>
```

Example 3

Define an ellipse with a horizontal linear gradient from yellow to red, and add a text inside the ellipse:



Here is the SVG code:

Example

```
<svg height="150" width="400">
  <defs>
    <linearGradient id="grad3" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad3)" />
  <text fill="#ffffff" font-size="45" font-family="Verdana" x="150" y="86">
    SVG</text>
</svg>
```

Code explanation:

- The `<text>` element is used to add a text

SVG Gradients - Radial

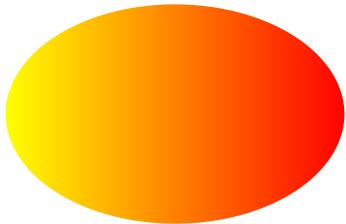
SVG Radial Gradient - <radialGradient>

The <radialGradient> element is used to define a radial gradient.

The <radialGradient> element must be nested within a <defs> tag. The <defs> tag is short for definitions and contains definition of special elements (such as gradients).

Example 1

Define an ellipse with a radial gradient from white to blue:



Here is the SVG code:

Example

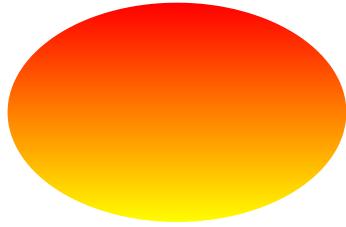
```
<svg height="150" width="500">
  <defs>
    <radialGradient id="grad1" cx="50%" cy="50%" r="50%" fx="50%" fy="50%">
      <stop offset="0%" style="stop-color:rgb(255,255,255); stop-opacity:0" />
      <stop offset="100%" style="stop-color:rgb(0,0,255);stop-opacity:1" />
    </radialGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

Code explanation:

- The id attribute of the <radialGradient> tag defines a unique name for the gradient
- The cx, cy and r attributes define the outermost circle and the fx and fy define the innermost circle
- The color range for a gradient can be composed of two or more colors. Each color is specified with a <stop> tag. The offset attribute is used to define where the gradient color begin and end
- The fill attribute links the ellipse element to the gradient

Example 2

Define another ellipse with a radial gradient from white to blue:



Here is the SVG code:

Example

```
<svg height="150" width="500">
  <defs>
    <radialGradient id="grad2" cx="20%" cy="30%" r="30%" fx="50%" fy="50%">
      <stop offset="0%" style="stop-color:rgb(255,255,255); stop-opacity:0" />
      <stop offset="100%" style="stop-color:rgb(0,0,255);stop-opacity:1" />
    </radialGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad2)" />
</svg>
```

SVG Examples

Try-it Yourself Examples

The examples below embed the SVG code directly into the HTML code.

This is supported by Firefox, Internet Explorer 9, Google Chrome, Opera, and Safari.

SVG Examples

SVG Basic Shapes

[A circle](#) [A rectangle](#) [A rectangle with opacity](#) [A rectangle with opacity 2](#) [A rectangle with rounded corners](#) [An ellipse](#) [Three ellipses on top of each other](#) [Two ellipses](#) [A line](#) [A polygon with three sides](#) [A polygon with four sides](#) [A star](#) [Another star](#) [A polyline](#) [Another polyline](#) [A path](#) [A quadratic Bézier curve](#) [Write a text](#) [Rotate a text](#) [Text on several lines](#) [Text as a link](#) [Defines the color of a line, text or outline \(stroke\)](#) [Defines the width of a line, text or outline \(stroke-width\)](#) [Defines different types of endings to an open path \(stroke-linecap\)](#) [Defines dashed lines \(stroke-dasharray\)](#)

SVG Filters

[feGaussianBlur - blur effect](#) [feOffset - offset a rectangle, then blend the original on top of the offset image](#) [feOffset - blur the offset image](#) [feOffset - make the shadow black](#) [feOffset - treat the shadow as a color](#) [A feBlend filter](#) [Filter 1](#) [Filter 2](#) [Filter 3](#) [Filter 4](#) [Filter 5](#) [Filter 6](#)

SVG Gradients

[An ellipse with a horizontal linear gradient from yellow to red](#) [An ellipse with a vertical linear gradient from yellow to red](#) [An ellipse with a horizontal linear gradient from yellow to red, and a text](#) [An ellipse with a radial gradient from white to blue](#) [Another ellipse with a radial gradient from white to blue](#)

SVG Misc

[Rectangle that repeatedly fade away over 5 seconds](#) [A growing rectangle that will change color](#) [Three rectangles that will change color](#) [Move text along a motion path](#) [Move, rotate, and scale text along a motion path](#) [Move, rotate, and scale text along a motion path + a growing rectangle that will change color](#) [Rotating ellipses](#)

SVG Reference

SVG Elements

Element	Description	Attributes
<a>	Creates a link around SVG elements	xlink:show xlink:actuate xlink:href target
<altGlyph>	Provides control over the glyphs used to render particular character data	x y dx dy rotate glyphRef format xlink:href
<altGlyphDef>	Defines a substitution set for glyphs	id
<altGlyphItem>	Defines a candidate set of glyph substitutions	id
<animate>	Defines how an attribute of an element changes over time	attributeName="the name of the target attribute" by="a relative offset value" from="the starting value" to="the ending value" dur="the duration" repeatCount="the number of time the animation will take place"
<animateMotion>	Causes a referenced element to move along a motion path	calcMode="the interpolation mode for the animation. Can be 'discrete', 'linear', 'paced', 'spline'" path="the motion path" keyPoints="how far along the motion path the object shall move at the moment in time" rotate="applies a rotation transformation" xlink:href="an URI reference to the <path> element which defines the motion path"
<animateTransform>	Animates a transformation attribute on a target element, thereby allowing animations to control translation, scaling, rotation and/or skewing	by="a relative offset value" from="the starting value" to="the ending value" type="the type of transformation which is to have its values change over time. Can be 'translate', 'scale', 'rotate', 'skewX', 'skewY'"

<circle>	Defines a circle	<code>cx="the x-axis center of the circle" cy="the y-axis center of the circle" r="The circle's radius". Required. + presentation attributes: Color, FillStroke, Graphics</code>
<clipPath>	Clipping is about hiding what normally would be drawn. The stencil which defines what is and what isn't drawn is called a clipping path	<code>clip-path="the referenced clipping path is intersected with the referencing clipping path" clipPathUnits="'userSpaceOnUse' or 'objectBoundingBox'. The second value makes units of children a fraction of the object bounding box which uses the mask (default: 'userSpaceOnUse')"</code>
<color-profile>	Specifies a color profile description (when the document is styled using CSS)	<code>local="the unique ID for a locally stored color profile" name="" rendering-intent="auto perceptual relative-colorimetric saturation absolute-colorimetric" xlink:href="the URI of an ICC profile resource"</code>
<cursor>	Defines a platform-independent custom cursor	<code>x="the x-axis top-left corner of the cursor (default is 0)" y="the y-axis top-left corner of the cursor (default is 0)" xlink:href="the URI of the image to use as the cursor"</code>
<defs>	A container for referenced elements	
<desc>	A text-only description for container elements or graphic elements in SVG (user agents may display the text as a tooltip)	
<ellipse>	Defines an ellipse	<code>cx="the x-axis center of the ellipse" cy="the y-axis center of the ellipse" rx="the length of the ellipse's radius along the x-axis". Required. ry="the length of the ellipse's radius along the y-axis". Required. + presentation attributes: Color, FillStroke, Graphics</code>
<feBlend>	Composes two objects together according to a	<code>mode="the image blending modes: normal multiply screen darken lighten" in="identifies input for the given filter primitive: SourceGraphic SourceAlpha </code>

	certain blending mode	BackgroundImage BackgroundAlpha FillPaint StrokePaint <filter-primitive-reference>" in2="the second input image to the blending operation"
feColorMatrix	SVG filter. Applies a matrix transformation	
feComponentTransfer	SVG filter. Performs component-wise remapping of data	
feComposite	SVG filter.	
feConvolveMatrix	SVG filter.	
feDiffuseLighting	SVG filter.	
feDisplacementMap	SVG filter.	
feDistantLight	SVG filter. Defines a light source	
feFlood	SVG filter.	
feFuncA	SVG filter. Sub-element to feComponentTransfer	
feFuncB	SVG filter. Sub-element to feComponentTransfer	
feFuncG	SVG filter. Sub-element to feComponentTransfer	
feFuncR	SVG filter. Sub-element to feComponentTransfer	
feGaussianBlur	SVG filter. Performs a Gaussian blur on the image	
feImage	SVG filter.	
feMerge	SVG filter. Creates image layers on top of each other	

feMergeNode	SVG filter. Sub-element to feMerge	
feMorphology	SVG filter. Performs a "fattening" or "thinning" on a source graphic	
feOffset	SVG filter. Moves an image relative to its current position	
fePointLight	SVG filter.	
feSpecularLighting	SVG filter.	
feSpotLight	SVG filter.	
feTile	SVG filter.	
feTurbulence	SVG filter.	
filter	Container for filter effects	
font	Defines a font	
font-face	Describes the characteristics of a font	
font-face-format		
font-face-name		
font-face-src		
font-face-uri		
foreignObject		
<g>	Used to group together elements	id="the name of the group" fill="the fill color for the group" opacity="the opacity for the group" + presentation attributes: All
glyph	Defines the graphics for a given glyph	
glyphRef	Defines a possible glyph to use	

hkern

<image>	Defines an image	x="the x-axis top-left corner of the image" y="the y-axis top-left corner of the image" width="the width of the image". Required. height="the height of the image". Required. xlink:href="the path to the image". Required. + presentation attributes: Color, Graphics, Images, Viewports
<line>	Defines a line	x1="the x start point of the line" y1="the y start point of the line" x2="the x end point of the line" y2="the y end point of the line" + presentation attributes: Color, FillStroke, Graphics, Markers
<linearGradient>	Defines a linear gradient. Linear gradients fill the object by using a vector, and can be defined as horizontal, vertical or angular gradients.	id="the unique id used to reference this pattern. Required to reference it" gradientUnits="'userSpaceOnUse' or 'objectBoundingBox'. Use the view box or object to determine relative position of vector points. (Default 'objectBoundingBox')" gradientTransform="the transformation to apply to the gradient" x1="the x start point of the gradient vector (number or % - 0% is default)" y1="the y start point of the gradient vector. (0% default)" x2="the x end point of the gradient vector. (100% default)" y2="the y end point of the gradient vector. (0% default)" spreadMethod="pad' or 'reflect' or 'repeat'" xlink:href="reference to another gradient whose attribute values are used as defaults and stops included. Recursive"
<marker>	Markers can be placed on the vertices of lines, polylines, polygons and paths. These elements can use the marker attributes "marker-start", "marker-mid" and "marker-end" which inherit by default or can be set to 'none' or the URI of a	markerUnits="strokeWidth' or 'userSpaceOnUse'. If 'strokeWidth' is used then one unit equals one stroke width. Otherwise, the marker does not scale and uses the the same view units as the referencing element (default 'strokeWidth')" refx="the position where the marker connects with the vertex (default 0)" refy="the position where the marker connects with the vertex (default 0)" orient="auto' or an angle to always show the marker at. 'auto' will compute an angle that makes the x-axis a tangent of the vertex

	defined marker. You must first define the marker before you can reference it via its URI. Any kind of shape can be put inside marker. They are drawn on top of the element they are attached to	(default 0)" markerWidth="the width of the marker (default 3)" markerHeight="the height of the marker (default 3)" viewBox="the points "seen" in this SVG drawing area. 4 values separated by white space or commas. (min x, min y, width, height)" + presentation attributes: All
<mask>	Masking is a combination of opacity values and clipping. Like clipping you can use shapes, text or paths to define sections of the mask. The default state of a mask is fully transparent which is the opposite of clipping plane. The graphics in a mask sets how opaque portions of the mask are	maskUnits=""userSpaceOnUse' or 'objectBoundingBox'. Set whether the clipping plane is relative the full view port or object (default: 'objectBoundingBox') maskContentUnits="Use the second with percentages to make mask graphic positions relative the object. 'userSpaceOnUse' or 'objectBoundingBox' (default: 'userSpaceOnUse')" x="the clipping plane of the mask (default: -10%)" y="the clipping plane of the mask (default: -10%)" width="the clipping plane of the mask (default: 120%)" height="the clipping plane of the mask (default: 120%)"
metadata	Specifies metadata	
missing-glyph		
mpath		
<path>	Defines a path	d="a set of commands which define the path" pathLength="If present, the path will be scaled so that the computed path length of the points equals this value" transform="a list of transformations" + presentation attributes: Color, FillStroke, Graphics, Markers
<pattern>	Defines the coordinates you want the view to show and the size of the view. Then you add shapes into your pattern. The pattern repeats	id="the unique id used to reference this pattern." Required. patternUnits=""userSpaceOnUse' or 'objectBoundingBox'. The second value makes units of x, y, width, height a fraction (or %) of the object bounding box which uses the pattern."

	when an edge of the view box (viewing area) is hit	patternContentUnits="''userSpaceOnUse' or 'objectBoundingBox'" patternTransform="allows the whole pattern to be transformed" x="pattern's offset from the top-left corner (default 0)" y="pattern's offset from the top-left corner. (default 0)" width="the width of the pattern tile (default 100%)" height="the height of the pattern tile (default 100%)" viewBox="the points "seen" in this SVG drawing area. 4 values separated by white space or commas. (min x, min y, width, height)" xlink:href="reference to another pattern whose attribute values are used as defaults and any children are inherited. Recursive"
<polygon>	Defines a graphic that contains at least three sides	points="the points of the polygon. The total number of points must be even". Required. fill-rule="part of the FillStroke presentation attributes" + presentation attributes: Color, FillStroke, Graphics, Markers
<polyline>	Defines any shape that consists of only straight lines	points="the points on the polyline". Required. + presentation attributes: Color, FillStroke, Graphics, Markers
<radialGradient>	Defines a radial gradient. Radial gradients are created by taking a circle and smoothly changing values between gradient stops from the focus point to the outside radius.	gradientUnits="''userSpaceOnUse' or 'objectBoundingBox'. Use the view box or object to determine relative position of vector points. (Default 'objectBoundingBox')" gradientTransform="the transformation to apply to the gradient" cx="the center point of the gradient (number or % - 50% is default)" cy="the center point of the gradient. (50% default)" r="the radius of the gradient. (50% default)" fx="the focus point of the gradient. (0% default)" fy="The focus point of the gradient. (0% default)" spreadMethod="pad' or 'reflect' or 'repeat'" xlink:href="Reference to another gradient whose attribute values are used as defaults and stops included. Recursive"
<rect>	Defines a rectangle	x="the x-axis top-left corner of the rectangle" y="the y-axis top-left corner of the rectangle" rx="the x-axis radius (to round the element)" ry="the y-axis radius (to round the element)" width="the width of

the rectangle". Required. height="the height of the rectangle" Required. + presentation attributes: Color, FillStroke, Graphics

script Container for scripts (e.g., ECMAScript)

set Sets the value of an attribute for a specified duration

<stop> The stops for a gradient offset="the offset for this stop (0 to 1/0% to 100%)". Required. stop-color="the color of this stop" stop-opacity="the opacity of this stop (0 to 1)"

style Allows style sheets to be embedded directly within SVG content

<svg> Creates an SVG document fragment x="top left corner when embedded (default 0)" y="top left corner when embedded (default 0)" width="the width of the svg fragment (default 100%)" height="the height of the svg fragment (default 100%)" viewBox="the points "seen" in this SVG drawing area. 4 values separated by white space or commas. (min x, min y, width, height)" preserveAspectRatio=""none' or any of the 9 combinations of 'xVALYVAL' where VAL is 'min', 'mid' or 'max'. (default xMidYMid)" zoomAndPan=""magnify' or 'disable'. Magnify option allows users to pan and zoom your file (default magnify)" xml="outermost <svg> element needs to setup SVG and its namespace: xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" xml:space="preserve"" + presentation attributes: All

switch

symbol

<text> Defines a text x="a list of x-axis positions. The nth x-axis position is given to the nth character in the text. If there are additional characters after

the positions run out they are placed after the last character. 0 is default" y="a list of y-axis positions. (see x). 0 is default" dx="a list of lengths which moves the characters relative to the absolute position of the last glyph drawn. (see x)" dy="a list of lengths which moves the characters relative to the absolute position of the last glyph drawn. (see x)" rotate="a list of rotations. The nth rotation is performed on the nth character. Additional characters are NOT given the last rotation value" textLength="a target length for the text that the SVG viewer will attempt to display the text between by adjusting the spacing and/or the glyphs. (default: The text's normal length)" lengthAdjust="tells the viewer what to adjust to try to accomplish rendering the text if the length is specified. The two values are 'spacing' and 'spacingAndGlyphs'" + presentation attributes: Color, FillStroke, Graphics, FontSpecification, TextContentElements

textPath

title	A text-only description for elements in SVG - not displayed as part of the graphics. User agents may display the text as a tooltip	
<tref>	References any <text> element in the SVG document and reuse it	Identical to the <text> element
<tspan>	Identical to the <text> element but can be nested inside text tags and inside itself	Identical to the <text> element + in addition: xlink:href="Reference to a <text> element"
<use>	Uses a URI to reference a <g>, <svg> or other graphical element	x="the x-axis top-left corner of the cloned element" y="the y-axis top-left corner of the cloned element" width="the width of the cloned element" height="the height of the

with a unique id attribute and replicate it. The copy is only a reference to the original so only the original exists in the document. Any change to the original affects all copies.

"cloned element" xlink:href="a URI reference to the cloned element" + presentation attributes: All

view

vkern



HTML Canvas Tutorial



The HTML <canvas> element is used to draw graphics on a web page.

The graphic above is created with <canvas>.

It shows four elements: a red rectangle, a gradient rectangle, a multicolor rectangle, and a multicolor text.

What is HTML Canvas?

The HTML <canvas> element is used to draw graphics, on the fly, via scripting (usually JavaScript).

The <canvas> element is only a container for graphics. You must use a script to actually draw the graphics.

Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

Browser Support

The numbers in the table specify the first browser version that fully supports the <canvas> element.

Element					
<canvas>	4.0	9.0	2.0	3.1	9.0

HTML Canvas Can Draw Text

Canvas can draw colorful text, with or without animation.

HTML Canvas Can Draw Graphics

Canvas has great features for graphical data presentation with an imagery of graphs and charts.

HTML Canvas Can be Animated

Canvas objects can move. Everything is possible: from simple bouncing balls to complex animations.

HTML Canvas Can be Interactive

Canvas can respond to JavaScript events.

Canvas can respond to any user action (key clicks, mouse clicks, button clicks, finger movement).

HTML Canvas Can be Used in Games

Canvas' methods for animations, offer a lot of possibilities for HTML gaming applications.

Canvas Example

In HTML, a <canvas> element looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

The <canvas> element must have an id attribute so it can be referred to by JavaScript.

The width and height attribute is necessary to define the size of the canvas.

Tip: You can have multiple <canvas> elements on one HTML page.

By default, the <canvas> element has no border and no content.

To add a border, use a style attribute:

Example

```
<canvas id="myCanvas" width="200" height="100"  
style="border:1px solid #000000;">  
</canvas>
```

The next chapters show how to draw on the canvas.

HTML Canvas Drawing

Draw on the Canvas With JavaScript

All drawing on the HTML canvas must be done with JavaScript:

Example

```
<script>
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(0,0,150,75);
</script>
```

Step 1: Find the Canvas Element

First of all, you must find the <canvas> element.

This is done by using the HTML DOM method getElementById():

```
var canvas = document.getElementById("myCanvas");
```

Step 2: Create a Drawing Object

Secondly, you need a drawing object for the canvas.

The getContext() is a built-in HTML object, with properties and methods for drawing:

```
var ctx = canvas.getContext("2d");
```

Step 3: Draw on the Canvas

Finally, you can draw on the canvas.

Set the fill style of the drawing object to the color red:

```
ctx.fillStyle = "#FF0000";
```

The `fillStyle` property can be a CSS color, a gradient, or a pattern. The default `fillStyle` is black. The `fillRect(x,y,width,height)` method draws a rectangle, filled with the fill style, on the canvas:

```
ctx.fillRect(0,0,150,75);
```

HTML Canvas Coordinates

Canvas Coordinates

The HTML canvas is a two-dimensional grid.

The upper-left corner of the canvas has the coordinates (0,0)

In the previous chapter, you saw this method used: fillRect(0,0,150,75).

This means: Start at the upper-left corner (0,0) and draw a 150x75 pixels rectangle.

Coordinates Example

Mouse over the rectangle below to see its x and y coordinates:

X

Y

Draw a Line

To draw a straight line on a canvas, use the following methods:

- moveTo(x,y) - defines the starting point of the line
- lineTo(x,y) - defines the ending point of the line

To actually draw the line, you must use one of the "ink" methods, like stroke().

Example



Define a starting point in position (0,0), and an ending point in position (200,100). Then use the stroke() method to actually draw the line:

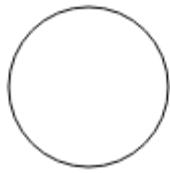
```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
```

Draw a Circle

To draw a circle on a canvas, use the following methods:

- `beginPath()` - begins a path
- `arc(x,y,r,startangle,endangle)` - creates an arc/curve. To create a circle with `arc()`: Set start angle to 0 and end angle to `2*Math.PI`. The x and y parameters define the x- and y-coordinates of the center of the circle. The r parameter defines the radius of the circle.

Example



Define a circle with the `arc()` method. Then use the `stroke()` method to actually draw the circle:

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
```

HTML Canvas Gradients

Canvas - Gradients

Gradients can be used to fill rectangles, circles, lines, text, etc. Shapes on the canvas are not limited to solid colors.

There are two different types of gradients:

- `createLinearGradient(x,y,x1,y1)` - creates a linear gradient
- `createRadialGradient(x,y,r,x1,y1,r1)` - creates a radial/circular gradient

Once we have a gradient object, we must add two or more color stops.

The `addColorStop()` method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.

To use the gradient, set the `fillStyle` or `strokeStyle` property to the gradient, then draw the shape (rectangle, text, or a line).

Using `createLinearGradient()`

Example

Create a linear gradient. Fill rectangle with the gradient:



JavaScript:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");

// Create gradient
var grd=ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);
```

Using createRadialGradient():

Example

Create a radial/circular gradient. Fill rectangle with the gradient:



JavaScript:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");

// Create gradient
var grd=ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
```

HTML Canvas Text

Drawing Text on the Canvas

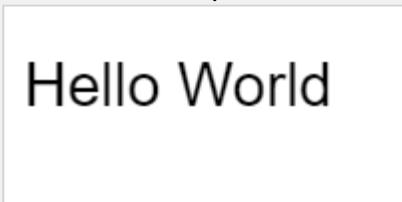
To draw text on a canvas, the most important property and methods are:

- font - defines the font properties for the text
- `fillText(text,x,y)` - draws "filled" text on the canvas
- `strokeText(text,x,y)` - draws text on the canvas (no fill)

Using `fillText()`

Example

Set font to 30px "Arial" and write a filled text on the canvas:



Hello World

JavaScript:

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.font = "30px Arial";
ctx.fillText("Hello World",10,50);
```

Using strokeText()

Example

Set font to 30px "Arial" and write a text, with no fill, on the canvas:



Hello World

JavaScript:

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.font = "30px Arial";
ctx.strokeText("Hello World", 10, 50);
```

Add Color and Center Text

Example

Set font to 30px "Comic Sans MS" and write a filled red text in the center of the canvas:



Hello World

JavaScript:

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.font = "30px Comic Sans MS";
ctx.fillStyle = "red";
ctx.TextAlign = "center";
ctx.fillText("Hello World", canvas.width/2, canvas.height/2);
```

HTML Canvas Images

Canvas - Images

To draw an image on a canvas, use the following method:

- `drawImage(image,x,y)`

Example



JavaScript:

```
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var ctx = canvas.getContext("2d");
    var img = document.getElementById("scream");
    ctx.drawImage(img, 10, 10);
};
```

HTML Canvas Reference

Description

The HTML5 <canvas> tag is used to draw graphics, on the fly, via scripting (usually JavaScript).

However, the <canvas> element has no drawing abilities of its own (it is only a container for graphics) - you must use a script to actually draw the graphics.

The getContext() method returns an object that provides methods and properties for drawing on the canvas.

This reference will cover the properties and methods of the getContext("2d") object, which can be used to draw text, lines, boxes, circles, and more - on the canvas.

Browser Support

The numbers in the table specify the first browser version that fully supports the element.

Element					
<canvas>	4.0	9.0	2.0	3.1	9.0

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support <canvas> and its properties and methods.

Note: Internet Explorer 8 and earlier versions, do not support the <canvas> element.

Colors, Styles, and Shadows

Property	Description
<u>fillStyle</u>	Sets or returns the color, gradient, or pattern used to fill the drawing
<u>strokeStyle</u>	Sets or returns the color, gradient, or pattern used for strokes
<u>shadowColor</u>	Sets or returns the color to use for shadows
<u>shadowBlur</u>	Sets or returns the blur level for shadows
<u>shadowOffsetX</u>	Sets or returns the horizontal distance of the shadow from the shape
<u>shadowOffsetY</u>	Sets or returns the vertical distance of the shadow from the shape

Method	Description
<u>createLinearGradient()</u>	Creates a linear gradient (to use on canvas content)
<u>createPattern()</u>	Repeats a specified element in the specified direction
<u>createRadialGradient()</u>	Creates a radial/circular gradient (to use on canvas content)
<u>addColorStop()</u>	Specifies the colors and stop positions in a gradient object

Line Styles

Property	Description
<u>lineCap</u>	Sets or returns the style of the end caps for a line
<u>lineJoin</u>	Sets or returns the type of corner created, when two lines meet
<u>lineWidth</u>	Sets or returns the current line width
<u>miterLimit</u>	Sets or returns the maximum miter length

Rectangles

Method	Description
<u>rect()</u>	Creates a rectangle
<u>fillRect()</u>	Draws a "filled" rectangle
<u>strokeRect()</u>	Draws a rectangle (no fill)
<u>clearRect()</u>	Clears the specified pixels within a given rectangle

Paths

Method	Description
<u>fill()</u>	Fills the current drawing (path)
<u>stroke()</u>	Actually draws the path you have defined
<u>beginPath()</u>	Begins a path, or resets the current path
<u>moveTo()</u>	Moves the path to the specified point in the canvas, without creating a line
<u>closePath()</u>	Creates a path from the current point back to the starting point
<u>lineTo()</u>	Adds a new point and creates a line to that point from the last specified point in the canvas
<u>clip()</u>	Clips a region of any shape and size from the original canvas
<u>quadraticCurveTo()</u>	Creates a quadratic Bézier curve
<u>bezierCurveTo()</u>	Creates a cubic Bézier curve
<u>arc()</u>	Creates an arc/curve (used to create circles, or parts of circles)
<u>arcTo()</u>	Creates an arc/curve between two tangents
<u>isPointInPath()</u>	Returns true if the specified point is in the current path, otherwise false

Transformations

Method	Description
<u>scale()</u>	Scales the current drawing bigger or smaller
<u>rotate()</u>	Rotates the current drawing
<u>translate()</u>	Remaps the (0,0) position on the canvas
<u>transform()</u>	Replaces the current transformation matrix for the drawing
<u>setTransform()</u>	Resets the current transform to the identity matrix. Then runs <u>transform()</u> .

Text

Property	Description
<u>font</u>	Sets or returns the current font properties for text content
<u>textAlign</u>	Sets or returns the current alignment for text content
<u>textBaseline</u>	Sets or returns the current text baseline used when drawing text

Method	Description
<u>fillText()</u>	Draws "filled" text on the canvas
<u>strokeText()</u>	Draws text on the canvas (no fill)
<u>measureText()</u>	Returns an object that contains the width of the specified text

Image Drawing

Method	Description
<u>drawImage()</u>	Draws an image, canvas, or video onto the canvas

Pixel Manipulation

Property	Description
<u>width</u>	Returns the width of an ImageData object
<u>height</u>	Returns the height of an ImageData object
<u>data</u>	Returns an object that contains image data of a specified ImageData object

Method	Description
<u>createImageData()</u>	Creates a new, blank ImageData object
<u>getImageData()</u>	Returns an ImageData object that copies the pixel data for the specified rectangle on a canvas
<u>putImageData()</u>	Puts the image data (from a specified ImageData object) back onto the canvas

Compositing

Property	Description
<u>globalAlpha</u>	Sets or returns the current alpha or transparency value of the drawing
<u>globalCompositeOperation</u>	Sets or returns how a new image are drawn onto an existing image

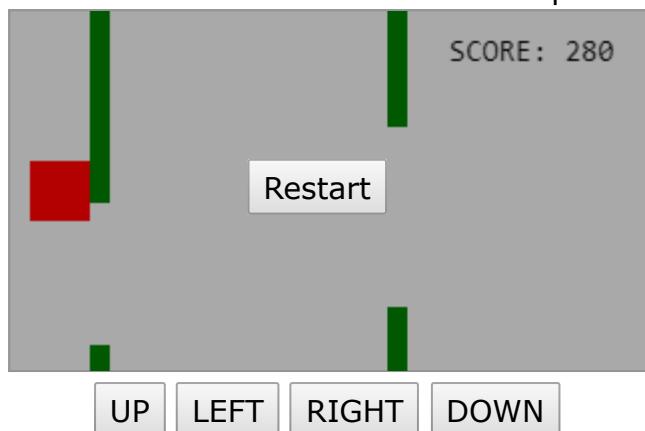
Other

Method	Description
<u>save()</u>	Saves the state of the current context
<u>restore()</u>	Returns previously saved path state and attributes
<u>createEvent()</u>	
<u>getContext()</u>	
<u>toDataURL()</u>	

HTML Game Example

Learn how to make games, using nothing but HTML and JavaScript.

Push the buttons to move the red square:



Try it Yourself Examples

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

Example

```
function startGame() {
    myGamePiece = new component(30, 30, "red", 10, 120);
    myGamePiece.gravity = 0.05;
    myScore = new component("30px", "Consolas", "black", 280, 40, "text");
    myGameArea.start();
}

var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.frameNo = 0;
    },
    clear : function() {
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    }
}
```

Game Canvas

The HTML `<canvas>` element is displayed as a rectangular object on a web page:

You can do anything

on the canvas

The `<canvas>` element is perfect for making games in HTML.

The `<canvas>` element offers all the functionality you need for making games.

Use JavaScript to draw, write, insert images, and more, onto the `<canvas>`.

`.getContext("2d")`

The `<canvas>` element has a built-in object, called the `getContext("2d")` object, with methods and properties for drawing.

You can learn more about the `<canvas>` element, and the `getContext("2d")` object, in our [Canvas Tutorial](#).

Get Started

To make a game, start by creating a gaming area, and make it ready for drawing:

Example

```
function startGame() {  
    myGameArea.start();  
}  
  
var myGameArea = {  
    canvas : document.createElement("canvas"),  
    start : function() {  
        this.canvas.width = 480;  
        this.canvas.height = 270;  
        this.context = this.canvas.getContext("2d");  
        document.body.insertBefore(this.canvas,  
document.body.childNodes[0]);  
    }  
}
```

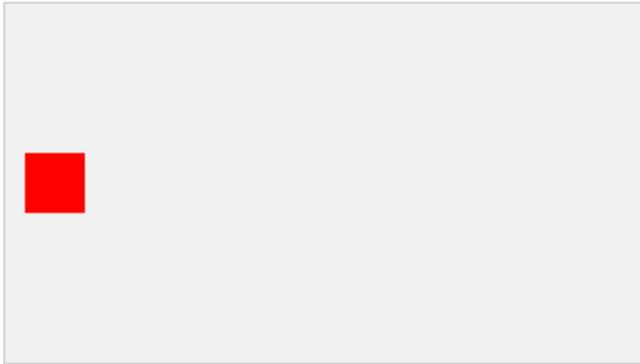
The object `myGameArea` will have more properties and methods later in this tutorial.

The function `startGame()` invokes the method `start()` of the `myGameArea` object.

The `start()` method creates a `<canvas>` element and inserts it as the first childnode of the `<body>` element.

Game Components

Add a red square onto the game area:



Add a Component

Make a component constructor, which lets you add components onto the gamearea. The object constructor is called `component`, and we make our first component, called `myGamePiece`:

Example

```
var myGamePiece;

function startGame() {
    myGameArea.start();
    myGamePiece = new component(30, 30, "red", 10, 120);
}

function component(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.x = x;
    this.y = y;
    ctx = myGameArea.context;
    ctx.fillStyle = color;
    ctx.fillRect(this.x, this.y, this.width, this.height);
}
```

The components have properties and methods to control their appearances and movements.

Frames

To make the game ready for action, we will update the display 50 times per second, which is much like frames in a movie.

First, create a new function called `updateGameArea()`.

In the `myGameArea` object, add an interval which will run the `updateGameArea()` function every 20th millisecond (50 times per second). Also add a function called `clear()`, that clears the entire canvas.

In the `component` constructor, add a function called `update()`, to handle the drawing of the component.

The `updateGameArea()` function calls the `clear()` and the `update()` method.

The result is that the component is drawn and cleared 50 times per second:

Example

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.interval = setInterval(updateGameArea, 20);
    },
    clear : function() {
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    }
}

function component(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.x = x;
    this.y = y;
    this.update = function(){
        ctx = myGameArea.context;
        ctx.fillStyle = color;
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }
}

function updateGameArea() {
    myGameArea.clear();
    myGamePiece.update();
}
```

Make it Move

To prove that the red square is being drawn 50 times per second, we will change the x position (horizontal) by one pixel every time we update the game area:

Example

```
function updateGameArea() {  
    myGameArea.clear();  
    myGamePiece.x += 1;  
    myGamePiece.update();  
}
```

Why Clear The Game Area?

It might seem unnecessary to clear the game area at every update. However, if we leave out the `clear()` method, all movements of the component will leave a trail of where it was positioned in the last frame:

Example

```
function updateGameArea() {  
    //myGameArea.clear();  
    myGamePiece.x += 1;  
    myGamePiece.update();  
}
```

Change the Size

You can control the width and height of the component:

Example

Create a 10x140 pixels rectangle:

```
function startGame() {  
    myGameArea.start();  
    myGamePiece = new component(10, 140, "red", 10, 120);  
}
```

Change the Color

You can control the color of the component:

Example

```
function startGame() {  
    myGameArea.start();  
    myGamePiece = new component(30, 30, "blue", 10, 120);  
}
```

You can also use other colorvalues like hex, rgb, or rgba:

Example

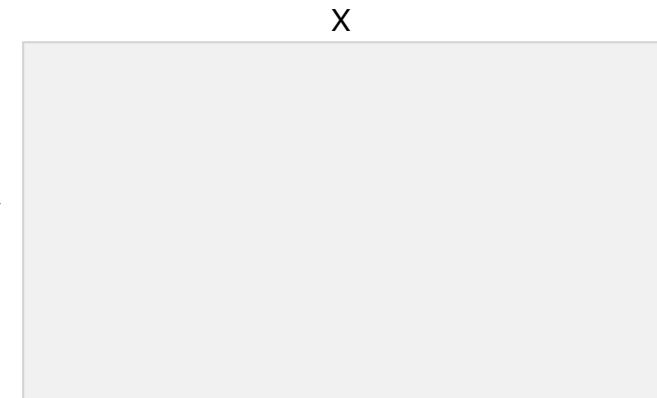
```
function startGame() {  
    myGameArea.start();  
    myGamePiece = new component(30, 30, "rgba(0, 0, 255, 0.5)", 10, 120);  
}
```

Change the Position

We use x- and y-coordinates to position components onto the game area.

The upper-left corner of the canvas has the coordinates (0,0)

Mouse over the game area below to see its x and y coordinates:



You can position the components wherever you like on the game area:

Example

```
function startGame() {  
    myGameArea.start();  
    myGamePiece = new component(30, 30, "red", 2, 2);  
}
```

Many Components

You can put as many components as you like on the game area:

Example

```
var redGamePiece, blueGamePiece, yellowGamePiece;

function startGame() {
    redGamePiece = new component(75, 75, "red", 10, 10);
    yellowGamePiece = new component(75, 75, "yellow", 50, 60);
    blueGamePiece = new component(75, 75, "blue", 10, 110);
    myGameArea.start();
}

function updateGameArea() {
    myGameArea.clear();
    redGamePiece.update();
    yellowGamePiece.update();
    blueGamePiece.update();
}
```

Moving Components

Make all three components move in different directions:

Example

```
function updateGameArea() {
    myGameArea.clear();
    redGamePiece.x += 1;
    yellowGamePiece.x += 1;
    yellowGamePiece.y += 1;
    blueGamePiece.x += 1;
    blueGamePiece.y -= 1;
    redGamePiece.update();
    yellowGamePiece.update();
    blueGamePiece.update();
}
```

Game Controllers

Push the buttons to move the red square:



Get in Control

Now we want to control the red square.

Add four buttons, up, down, left, and right.

Write a function for each button to move the component in the selected direction.

Make two new properties in the `component` constructor, and call them `speedX` and `speedY`.

These properties are being used as speed indicators.

Add a function in the `component` constructor, called `newPos()`, which uses the `speedX` and `speedY` properties to change the component's position.

The `newpos` function is called from the `updateGameArea` function before drawing the component:

Example

```
<script>
function component(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.fillStyle = color;
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }
    this.newPos = function() {
        this.x += this.speedX;
    }
}
```

```
    this.y += this.speedY;
}

}

function updateGameArea() {
    myGameArea.clear();
    myGamePiece.newPos();
    myGamePiece.update();
}

function moveup() {
    myGamePiece.speedY -= 1;
}

function movedown() {
    myGamePiece.speedY += 1;
}

function moveleft() {
    myGamePiece.speedX -= 1;
}

function moveright() {
    myGamePiece.speedX += 1;
}
</script>

<button onclick="moveup()">UP</button>
<button onclick="movedown()">DOWN</button>
<button onclick="moveleft()">LEFT</button>
<button onclick="moveright()">RIGHT</button>
```

Stop Moving

If you want, you can make the red square stop when you release a button.

Add a function that will set the speed indicators to 0.

To deal with both normal screens and touch screens, we will add code for both devices:

Example

```
function stopMove() {  
    myGamePiece.speedX = 0;  
    myGamePiece.speedY = 0;  
}  
</script>  
  
<button onmousedown="moveup()" onmouseup="stopMove()"  
ontouchstart="moveup()>UP</button>  
<button onmousedown="movedown()" onmouseup="stopMove()"  
ontouchstart="movedown()>DOWN</button>  
<button onmousedown="moveleft()" onmouseup="stopMove()"  
ontouchstart="moveleft()>LEFT</button>  
<button onmousedown="moveright()" onmouseup="stopMove()"  
ontouchstart="moveright()>RIGHT</button>
```

Keyboard as Controller

We can also control the red square by using the arrow keys on the keyboard.

Create a method that checks if a key is pressed, and set the `key` property of the `myGameArea` object to the key code. When the key is released, set the `key` property to `false`:

Example

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.interval = setInterval(updateGameArea, 20);
        window.addEventListener('keydown', function (e) {
            myGameArea.key = e.keyCode;
        })
        window.addEventListener('keyup', function (e) {
            myGameArea.key = false;
        })
    },
    clear : function(){
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    }
}
```

Then we can move the red square if one of the arrow keys are pressed:

Example

```
function updateGameArea() {  
    myGameArea.clear();  
    myGamePiece.speedX = 0;  
    myGamePiece.speedY = 0;  
    if (myGameArea.key && myGameArea.key == 37) {myGamePiece.speedX = -1; }  
    if (myGameArea.key && myGameArea.key == 39) {myGamePiece.speedX = 1; }  
    if (myGameArea.key && myGameArea.key == 38) {myGamePiece.speedY = -1; }  
    if (myGameArea.key && myGameArea.key == 40) {myGamePiece.speedY = 1; }  
    myGamePiece.newPos();  
    myGamePiece.update();  
}
```

Multiple Keys Pressed

What if more than one key is pressed at the same time?

In the example above, the component can only move horizontally or vertically. Now we want the component to also move diagonally.

Create a `keys` array for the `myGameArea` object, and insert one element for each key that is pressed, and give it the value `true`, the value remains true until the key is no longer pressed, the value becomes `false` in the `keyup` event listener function:

Example

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.interval = setInterval(updateGameArea, 20);
        window.addEventListener('keydown', function (e) {
            myGameArea.keys = (myGameArea.keys || []);
            myGameArea.keys[e.keyCode] = true;
        })
        window.addEventListener('keyup', function (e) {
            myGameArea.keys[e.keyCode] = false;
        })
    },
    clear : function(){
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    }
}

function updateGameArea() {
    myGameArea.clear();
    myGamePiece.speedX = 0;
    myGamePiece.speedY = 0;
    if (myGameArea.keys && myGameArea.keys[37]) {myGamePiece.speedX = -1; }
    if (myGameArea.keys && myGameArea.keys[39]) {myGamePiece.speedX = 1; }
    if (myGameArea.keys && myGameArea.keys[38]) {myGamePiece.speedY = -1; }
    if (myGameArea.keys && myGameArea.keys[40]) {myGamePiece.speedY = 1; }
```

```
    myGamePiece.newPos();
    myGamePiece.update();
}
```

Using The Mouse Cursor as a Controller

If you want to control the red square by using the mouse cursor, add a method in `myGameArea` object that updates the x and y coordinates of the mouse cursor:.

Example

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.canvas.style.cursor = "none"; //hide the original cursor
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.interval = setInterval(updateGameArea, 20);
        window.addEventListener('mousemove', function (e) {
            myGameArea.x = e.pageX;
            myGameArea.y = e.pageY;
        })
    },
    clear : function(){
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    }
}
```

Then we can move the red square using the mouse cursor:

Example

```
function updateGameArea() {  
    myGameArea.clear();  
    if (myGameArea.x && myGameArea.y) {  
        myGamePiece.x = myGameArea.x;  
        myGamePiece.y = myGameArea.y;  
    }  
    myGamePiece.update();  
}
```

Touch The Screen to Control The Game

We can also control the red square on a touch screen.

Add a method in the `myGameArea` object that uses the x and y coordinates of where the screen is touched:

Example

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.interval = setInterval(updateGameArea, 20);
        window.addEventListener('touchmove', function (e) {
            myGameArea.x = e.touches[0].screenX;
            myGameArea.y = e.touches[0].screenY;
        })
    },
    clear : function(){
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    }
}
```

Then we can move the red square if the user touches the screen, by using the same code as we did for the mouse cursor:

Example

```
function updateGameArea() {  
    myGameArea.clear();  
    if (myGameArea.touchX && myGameArea.touchY) {  
        myGamePiece.x = myGameArea.x;  
        myGamePiece.y = myGameArea.y;  
    }  
    myGamePiece.update();  
}
```

Controllers on The Canvas

We can also draw our own buttons on the canvas, and use them as controllers:

Example

```
function startGame() {  
    myGamePiece = new component(30, 30, "red", 10, 120);  
    myUpBtn = new component(30, 30, "blue", 50, 10);  
    myDownBtn = new component(30, 30, "blue", 50, 70);  
    myLeftBtn = new component(30, 30, "blue", 20, 40);  
    myRightBtn = new component(30, 30, "blue", 80, 40);  
    myGameArea.start();  
}
```

Add a new function that figures out if a component, in this case a button, is clicked. Start by adding event listeners to check if a mouse button is clicked (`mousedown` and `mouseup`). To deal with touch screens, also add event listeners to check if the screen is clicked on (`touchstart` and `touchend`):

Example

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.interval = setInterval(updateGameArea, 20);
        window.addEventListener('mousedown', function (e) {
            myGameArea.x = e.pageX;
            myGameArea.y = e.pageY;
        })
        window.addEventListener('mouseup', function (e) {
            myGameArea.x = false;
            myGameArea.y = false;
        })
        window.addEventListener('touchstart', function (e) {
            myGameArea.x = e.pageX;
            myGameArea.y = e.pageY;
        })
        window.addEventListener('touchend', function (e) {
            myGameArea.x = false;
            myGameArea.y = false;
        })
    },
    clear : function(){
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    }
}
```

Now the `myGameArea` object has properties that tells us the x- and y-coordinates of a click. We use these properties to check if the click was performed on one of our blue buttons. The new method is called `clicked`, it is a method of the `component` constructor, and it checks if the component is being clicked.

In the `updateGameArea` function, we take the neccessary actions if one of the blue buttons is clicked:

Example

```
function component(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.fillStyle = color;
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }
    this.clicked = function() {
        var myleft = this.x;
        var myright = this.x + (this.width);
        var mytop = this.y;
        var mybottom = this.y + (this.height);
        var clicked = true;
        if ((mybottom < myGameArea.y) || (mytop > myGameArea.y)
            || (myright < myGameArea.x) || (myleft > myGameArea.x)) {
            clicked = false;
        }
        return clicked;
    }
}

function updateGameArea() {
    myGameArea.clear();
    if (myGameArea.x && myGameArea.y) {
        if (myUpBtn.clicked()) {
            myGamePiece.y -= 1;
        }
        if (myDownBtn.clicked()) {
            myGamePiece.y += 1;
        }
    }
}
```

```
    }
    if (myLeftBtn.clicked()) {
        myGamePiece.x += -1;
    }
    if (myRightBtn.clicked()) {
        myGamePiece.x += 1;
    }
}
myUpBtn.update();
myDownBtn.update();
myLeftBtn.update();
myRightBtn.update();
myGamePiece.update();
}
```

Game Obstacles

Push the buttons to move the red square:



Add Some Obstacles

Now we want to add some obstacles to our game.

Add a new component to the gaming area. Make it green, 10px wide, 200px high, and place it 300px to the right and 120px down.

Also update the obstacle component in every frame:

Example

```
var myGamePiece;
var myObstacle;

function startGame() {
    myGamePiece = new component(30, 30, "red", 10, 120);
    myObstacle = new component(10, 200, "green", 300, 120);
    myGameArea.start();
}

function updateGameArea() {
    myGameArea.clear();
    myObstacle.update();
    myGamePiece.newPos();
    myGamePiece.update();
}
```

Hit The Obstacle = Game Over

In the example above, nothing happens when you hit the obstacle. In a game, that is not very satisfying.

How do we know if our red square hits the obstacle?

Create a new method in the component constructor, that checks if the component crashes with another component. This method should be called every time the frames updates, 50 times per second.

Also add a `stop()` method to the `myGameArea` object, which clears the 20 milliseconds interval.

Example

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.interval = setInterval(updateGameArea, 20);
    },
    clear : function() {
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    },
    stop : function() {
        clearInterval(this.interval);
    }
}

function component(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.fillStyle = color;
```

```

        ctx.fillRect(this.x, this.y, this.width, this.height);
    }

    this.newPos = function() {
        this.x += this.speedX;
        this.y += this.speedY;
    }

    this.crashWith = function(otherobj) {
        var myleft = this.x;
        var myright = this.x + (this.width);
        var mytop = this.y;
        var mybottom = this.y + (this.height);
        var otherleft = otherobj.x;
        var otherright = otherobj.x + (otherobj.width);
        var othertop = otherobj.y;
        var otherbottom = otherobj.y + (otherobj.height);
        var crash = true;
        if ((mybottom < othertop) ||
            (mytop > otherbottom) ||
            (myright < otherleft) ||
            (myleft > otherright)) {
            crash = false;
        }
        return crash;
    }

}

function updateGameArea() {
    if (myGamePiece.crashWith(myObstacle)) {
        myGameArea.stop();
    } else {
        myGameArea.clear();
        myObstacle.update();
        myGamePiece.newPos();
        myGamePiece.update();
    }
}

```

Moving Obstacle

The obstacle is of no danger when it is static, so we want it to move.

Change the property value of `myObstacle.x` at every update:

Example

```
function updateGameArea() {
    if (myGamePiece.crashWith(myObstacle)) {
        myGameArea.stop();
    } else {
        myGameArea.clear();
        myObstacle.x += -1;
        myObstacle.update();
        myGamePiece.newPos();
        myGamePiece.update();
    }
}
```

Multiple Obstacles

How about adding multiple obstacles?

For that we need a property for counting frames, and a method for execute something at a given frame rate.

Example

```
var myGameArea = {
    canvas : document.createElement("canvas"),
    start : function() {
        this.canvas.width = 480;
        this.canvas.height = 270;
        this.context = this.canvas.getContext("2d");
        document.body.insertBefore(this.canvas,
document.body.childNodes[0]);
        this.frameNo = 0;
        this.interval = setInterval(updateGameArea, 20);
    },
    clear : function() {
        this.context.clearRect(0, 0, this.canvas.width,
this.canvas.height);
    },
    stop : function() {
        clearInterval(this.interval);
    }
}

function everyinterval(n) {
    if ((myGameArea.frameNo / n) % 1 == 0) {return true;}
    return false;
}
```

The `everyinterval` function returns true if the current framenumber corresponds with the given interval.

To define multiple obstacles, first declare the `obstacle` variable as an array.

Second, we need to make some changes in the `updateGameArea` function.

Example

```
var myGamePiece;
var myObstacles = [];

function updateGameArea() {
    var x, y;
    for (i = 0; i < myObstacles.length; i += 1) {
        if (myGamePiece.crashWith(myObstacles[i])) {
            myGameArea.stop();
            return;
        }
    }
    myGameArea.clear();
    myGameArea.frameNo += 1;
    if (myGameArea.frameNo == 1 || everyinterval(150)) {
        x = myGameArea.canvas.width;
        y = myGameArea.canvas.height - 200
        myObstacles.push(new component(10, 200, "green", x, y));
    }
    for (i = 0; i < myObstacles.length; i += 1) {
        myObstacles[i].x += -1;
        myObstacles[i].update();
    }
    myGamePiece.newPos();
    myGamePiece.update();
}
```

In the `updateGameArea` function we must loop through every obstacle to see if there is a crash. If there is a crash, the `updateGameArea` function will stop, and no more drawing is done.

The `updateGameArea` function counts frames and adds an obstacle for every 150th frame.

Obstacles of Random Size

To make the game a bit more difficult, and fun, we will send in obstacles of random sizes, so that the red square must move up and down to not crash.

Example

```
function updateGameArea() {
    var x, height, gap, minHeight, maxHeight, minGap, maxGap;
    for (i = 0; i < myObstacles.length; i += 1) {
        if (myGamePiece.crashWith(myObstacles[i])) {
            myGameArea.stop();
            return;
        }
    }
    myGameArea.clear();
    myGameArea.frameNo += 1;
    if (myGameArea.frameNo == 1 || everyinterval(150)) {
        x = myGameArea.canvas.width;
        minHeight = 20;
        maxHeight = 200;
        height = Math.floor(Math.random()*(maxHeight-
minHeight+1)+minHeight);
        minGap = 50;
        maxGap = 200;
        gap = Math.floor(Math.random()*(maxGap-minGap+1)+minGap);
        myObstacles.push(new component(10, height, "green", x, 0));
        myObstacles.push(new component(10, x - height - gap, "green", x,
height + gap));
    }
    for (i = 0; i < myObstacles.length; i += 1) {
        myObstacles[i].x += -1;
        myObstacles[i].update();
    }
    myGamePiece.newPos();
    myGamePiece.update();
}
```

Game Score

Push the buttons to move the red square:



Count The Score

There are many ways to keep the score in a game, we will show you how to write a score onto the canvas.

First make a score component:

Example

```
var myGamePiece;
var myObstacles = [];
var myScore;

function startGame() {
    myGamePiece = new component(30, 30, "red", 10, 160);
    myScore = new component("30px", "Consolas", "black", 280, 40, "text");
    myGameArea.start();
}
```

The syntax for writing text on a canvas element is different from drawing a rectangle. Therefore we must call the component constructor using an additional argument, telling the constructor that this component is of type "text".

In the component constructor we test if the component is of type "text", and use the `fillText` method instead of the `fillRect` method:

Example

```
function component(width, height, color, x, y, type) {
    this.type = type;
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        if (this.type == "text") {
            ctx.font = this.width + " " + this.height;
            ctx.fillStyle = color;
            ctx.fillText(this.text, this.x, this.y);
        } else {
            ctx.fillStyle = color;
            ctx.fillRect(this.x, this.y, this.width, this.height);
        }
    }
    ...
}
```

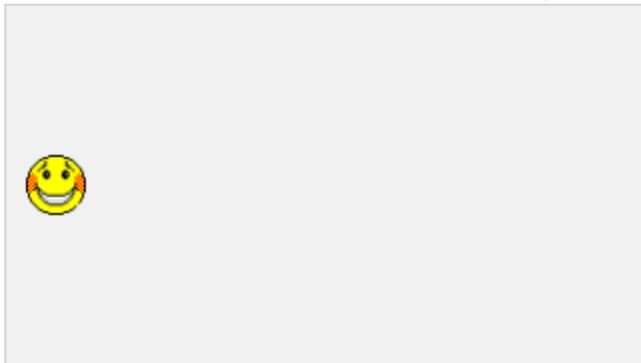
At last we add some code in the updateGameArea function that writes the score onto the canvas. We use the `frameNo` property to count the score:

Example

```
function updateGameArea() {  
    var x, height, gap, minHeight, maxHeight, minGap, maxGap;  
    for (i = 0; i < myObstacles.length; i += 1) {  
        if (myGamePiece.crashWith(myObstacles[i])) {  
            myGameArea.stop();  
            return;  
        }  
    }  
    myGameArea.clear();  
    myGameArea.frameNo += 1;  
    if (myGameArea.frameNo == 1 || everyinterval(150)) {  
        x = myGameArea.canvas.width;  
        minHeight = 20;  
        maxHeight = 200;  
        height = Math.floor(Math.random()*(maxHeight-  
minHeight+1)+minHeight);  
        minGap = 50;  
        maxGap = 200;  
        gap = Math.floor(Math.random()*(maxGap-minGap+1)+minGap);  
        myObstacles.push(new component(10, height, "green", x, 0));  
        myObstacles.push(new component(10, x - height - gap, "green", x,  
height + gap));  
    }  
    for (i = 0; i < myObstacles.length; i += 1) {  
        myObstacles[i].speedX = -1;  
        myObstacles[i].newPos();  
        myObstacles[i].update();  
    }  
    myScore.text="SCORE: " + myGameArea.frameNo;  
    myScore.update();  
    myGamePiece.newPos();  
    myGamePiece.update();  
}
```

Game Images

Push the buttons to move the smiley:



UP **LEFT** **RIGHT** **DOWN**

How to Use Images?

To add images on a canvas, the `getContext("2d")` object has built-in image properties and methods.

In our game, to create the gamepiece as an image, use the component constructor, but instead of referring to a color, you must refer to the url of the image. And you must tell the constructor that this component is of type "image":

Example

```
function startGame() {  
    myGamePiece = new component(30, 30, "smiley.gif", 10, 120, "image");  
    myGameArea.start();  
}
```

In the component constructor we test if the component is of type "image", and create an image object by using the built-in "new Image()" object constructor. When we are ready to draw the image, we use the drawImage method instead of the fillRect method:

Example

```
function component(width, height, color, x, y, type) {
    this.type = type;
    if (type == "image") {
        this.image = new Image();
        this.image.src = color;
    }
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        if (type == "image") {
            ctx.drawImage(this.image,
                this.x,
                this.y,
                this.width, this.height);
        } else {
            ctx.fillStyle = color;
            ctx.fillRect(this.x, this.y, this.width, this.height);
        }
    }
}
```

Change Images

You can change the image whenever you like by changing the `src` property of the `image` object of your component.



If you want to change the smiley everytime it moves, change the image source when the user clicks a button, and back to normal when the button is not clicked:

Example

```
function move(dir) {  
    myGamePiece.image.src = "angry.gif";  
    if (dir == "up") {myGamePiece.speedY = -1; }  
    if (dir == "down") {myGamePiece.speedY = 1; }  
    if (dir == "left") {myGamePiece.speedX = -1; }  
    if (dir == "right") {myGamePiece.speedX = 1; }  
}  
  
function clearmove() {  
    myGamePiece.image.src = "smiley.gif";  
    myGamePiece.speedX = 0;  
    myGamePiece.speedY = 0;  
}
```

Background Images

Add a background image to your game area by adding it as a component, and also update the background in every frame:

Example

```
var myGamePiece;
var myBackground;

function startGame() {
    myGamePiece = new component(30, 30, "smiley.gif", 10, 120, "image");
    myBackground = new component(656, 270, "citymarket.jpg", 0, 0,
"image");
    myGameArea.start();
}

function updateGameArea() {
    myGameArea.clear();
    myBackground.newPos();
    myBackground.update();
    myGamePiece.newPos();
    myGamePiece.update();
}
```

Moving Background

Change the background component's `speedX` property to make the background move:

Example

```
function updateGameArea() {
    myGameArea.clear();
    myBackground.speedX = -1;
    myBackground.newPos();
    myBackground.update();
    myGamePiece.newPos();
    myGamePiece.update();
}
```

Background Loop

To make the same background loop forever, we must use a specific technique.

Start by telling the component constructor that this is a *background*. The component constructor will then add the image twice, placing the second image immediately after the first image.

In the `newPos()` method, check if the `x` position of the component has reach the end of the image, if it has, set the `x` position of the component to 0:

Example

```
function component(width, height, color, x, y, type) {
    this.type = type;
    if (type == "image" || type == "background") {
        this.image = new Image();
        this.image.src = color;
    }
    this.width = width;
    this.height = height;
    this.speedX = 0;
    this.speedY = 0;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        if (type == "image" || type == "background") {
            ctx.drawImage(this.image,
                this.x, this.y, this.width, this.height);
            if (type == "background") {
                ctx.drawImage(this.image,
                    this.x + this.width, this.y, this.width, this.height);
            }
        } else {
            ctx.fillStyle = color;
            ctx.fillRect(this.x, this.y, this.width, this.height);
        }
    }
    this.newPos = function() {
        this.x += this.speedX;
        this.y += this.speedY;
        if (this.type == "background") {
            if (this.x == -(this.width)) {
                this.x = 0;
            }
        }
    }
}
```

Game Sound

Turn up the volume. Do you hear a "dunk" when the red square hits an obstacle?



How to Add Sounds?

Use the HTML5 `<audio>` element to add sound and music to your games.

In our examples, we create a new object constructor to handle sound objects:

Example

```
function sound(src) {  
    this.sound = document.createElement("audio");  
    this.sound.src = src;  
    this.sound.setAttribute("preload", "auto");  
    this.sound.setAttribute("controls", "none");  
    this.sound.style.display = "none";  
    document.body.appendChild(this.sound);  
    this.play = function(){  
        this.sound.play();  
    }  
    this.stop = function(){  
        this.sound.pause();  
    }  
}
```

To create a new sound object use the `sound` constructor, and when the red square hits an obstacle, play the sound:

Example

```
var myGamePiece;
var myObstacles = [];
var mySound;

function startGame() {
    myGamePiece = new component(30, 30, "red", 10, 120);
    mySound = new sound("bounce.mp3");
    myGameArea.start();
}

function updateGameArea() {
    var x, height, gap, minHeight, maxHeight, minGap, maxGap;
    for (i = 0; i < myObstacles.length; i += 1) {
        if (myGamePiece.crashWith(myObstacles[i])) {
            mySound.play();
            myGameArea.stop();
            return;
        }
    }
}

...
```

Background Music

To add background music to your game, add a new sound object, and start playing when you start the game:

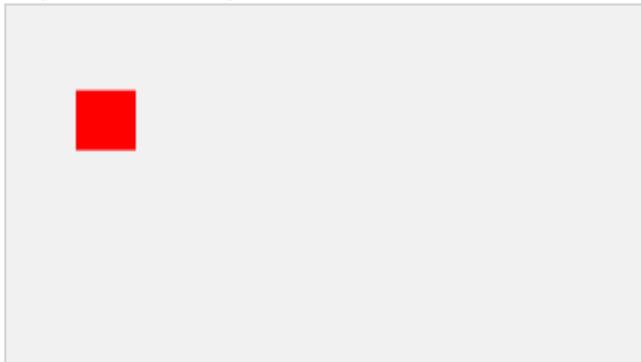
Example

```
var myGamePiece;
var myObstacles = [];
var mySound;
var myMusic;

function startGame() {
    myGamePiece = new component(30, 30, "red", 10, 120);
    mySound = new sound("bounce.mp3");
    myMusic = new sound("gametheme.mp3");
    myMusic.play();
    myGameArea.start();
}
```

Game Gravity

Some games have forces that pulls the game component in one direction, like gravity pulls objects to the ground.



RESTART

To add this functionality to our component constructor, first add a `gravity` property, which sets the current gravity. Then add a `gravitySpeed` property, which increases everytime we update the frame:

Example

```
function component(width, height, color, x, y, type) {
    this.type = type;
    this.width = width;
    this.height = height;
    this.x = x;
    this.y = y;
    this.speedX = 0;
    this.speedY = 0;
    this.gravity = 0.05;
    this.gravitySpeed = 0;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.fillStyle = color;
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }
    this.newPos = function() {
        this.gravitySpeed += this.gravity;
        this.x += this.speedX;
        this.y += this.speedY + this.gravitySpeed;
    }
}
```

Hit the Bottom

To prevent the red square from falling forever, stop the falling when it hits the bottom of the game area:

Example

```
this.newPos = function() {
    this.gravitySpeed += this.gravity;
    this.x += this.speedX;
    this.y += this.speedY + this.gravitySpeed;
    this.hitBottom();
}
this.hitBottom = function() {
    var rockbottom = myGameArea.canvas.height - this.height;
    if (this.y > rockbottom) {
        this.y = rockbottom;
    }
}
```

Accelerate Up

In a game, when you have a force that pulls you down, you should have a method to force the component to accelerate up.

Example

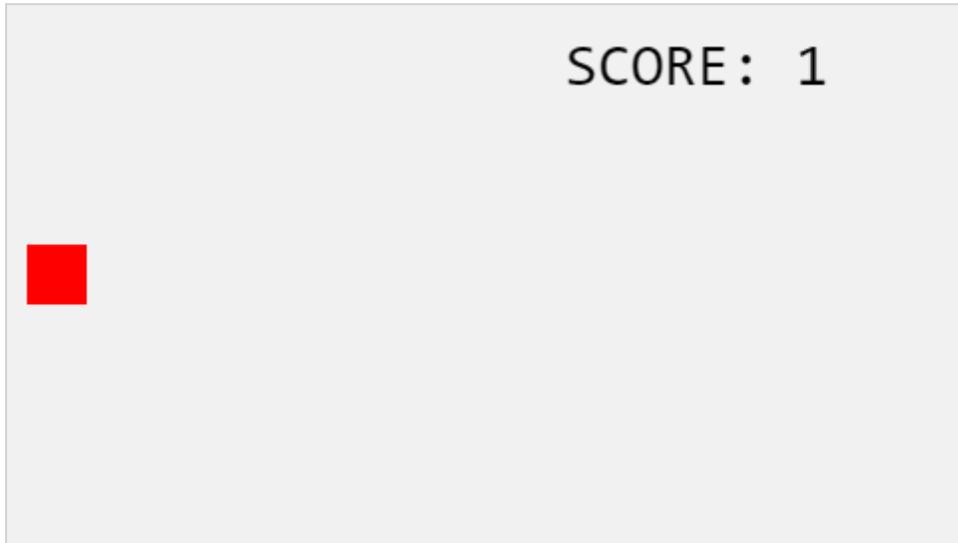
```
<script>
class="jsHigh">function accelerate(n) {
    myGamePiece.gravity = n;
}
</script>

<button onmousedown="accelerate(-0.2)"
onmouseup="accelerate(0.1)">ACCELERATE</button>
```

A Game

Make a game based on what we have learned so far:

Example



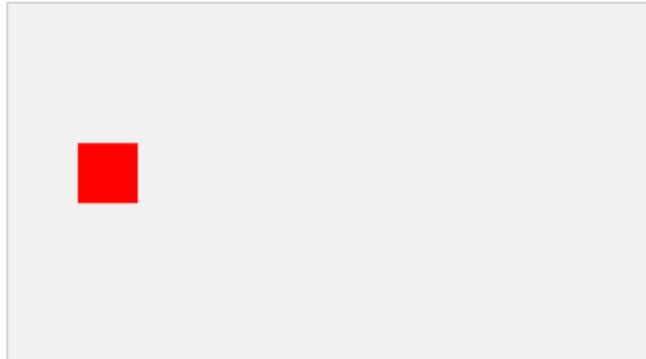
ACCELERATE

Click the ACCELERATE button to start the game

How long can you stay alive? Use the ACCELERATE button to stay in the air..

Game Bouncing

This red square bounces when it hits the floor:



RESTART

Bouncing

Another functionality we want to add is the `bounce` property.

The `bounce` property indicates if the component will bounce back when gravity makes it fall down to the ground.

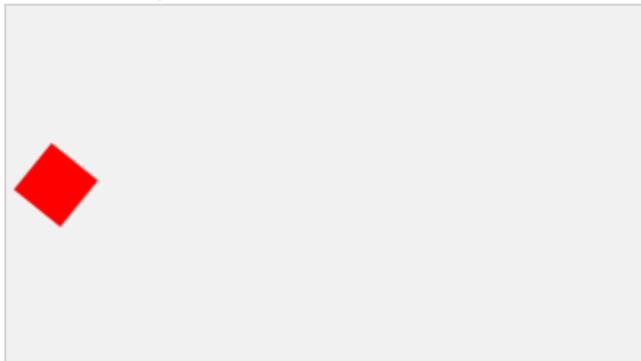
The bounce property value must be a number. 0 is no bounce at all, and 1 will make the component bounce all the way back to where it started falling.

Example

```
function component(width, height, color, x, y, type) {
    this.type = type;
    this.width = width;
    this.height = height;
    this.x = x;
    this.y = y;
    this.speedX = 0;
    this.speedY = 0;
    this.gravity = 0.1;
    this.gravitySpeed = 0;
    this.bounce = 0.6;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.fillStyle = color;
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }
    this.newPos = function() {
        this.gravitySpeed += this.gravity;
        this.x += this.speedX;
        this.y += this.speedY + this.gravitySpeed;
        this.hitBottom();
    }
    this.hitBottom = function() {
        var rockbottom = this.gamearea.canvas.height - this.height;
        if (this.y > rockbottom) {
            this.y = rockbottom;
            this.gravitySpeed = -(this.gravitySpeed * this.bounce);
        }
    }
}
```

Game Rotation

The red square can rotate:



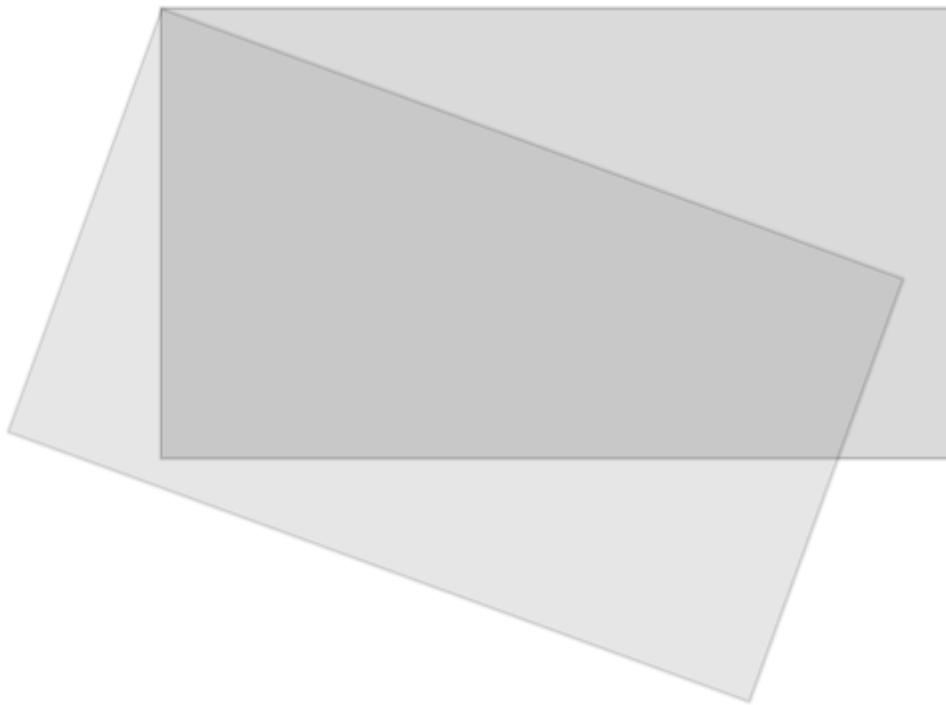
Rotate

Rotating Components

Earlier in this tutorial, the red square was able to move around on the gamearea, but it could not turn or rotate.

To rotate components, we have to change the way we draw components.

The only rotation method available for the canvas element will rotate the entire canvas:



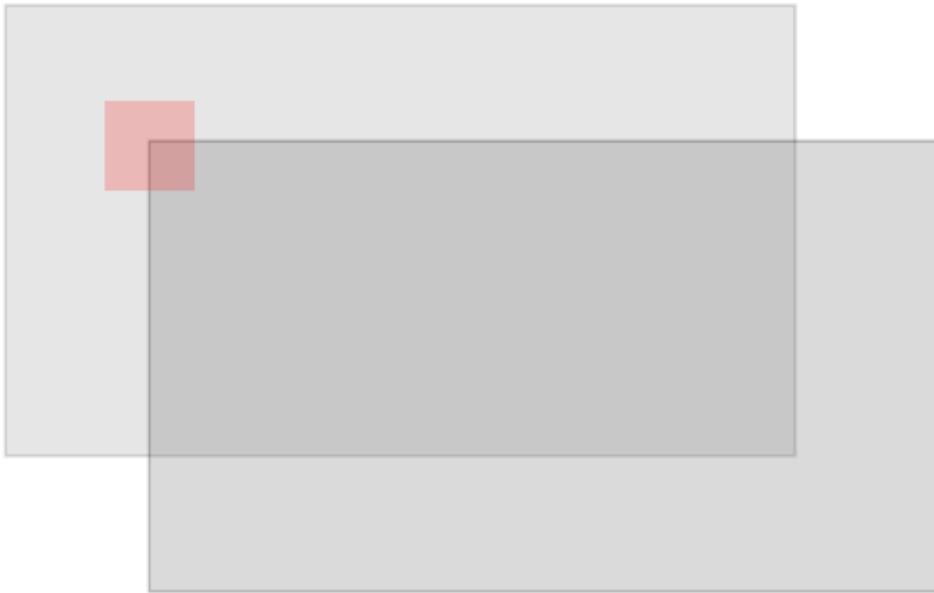
Everything else you draw on the canvas will also be rotated, not only the specific component. That is why we have to make some changes in the `update()` method:

First, we save the current canvas context object:

```
ctx.save();
```

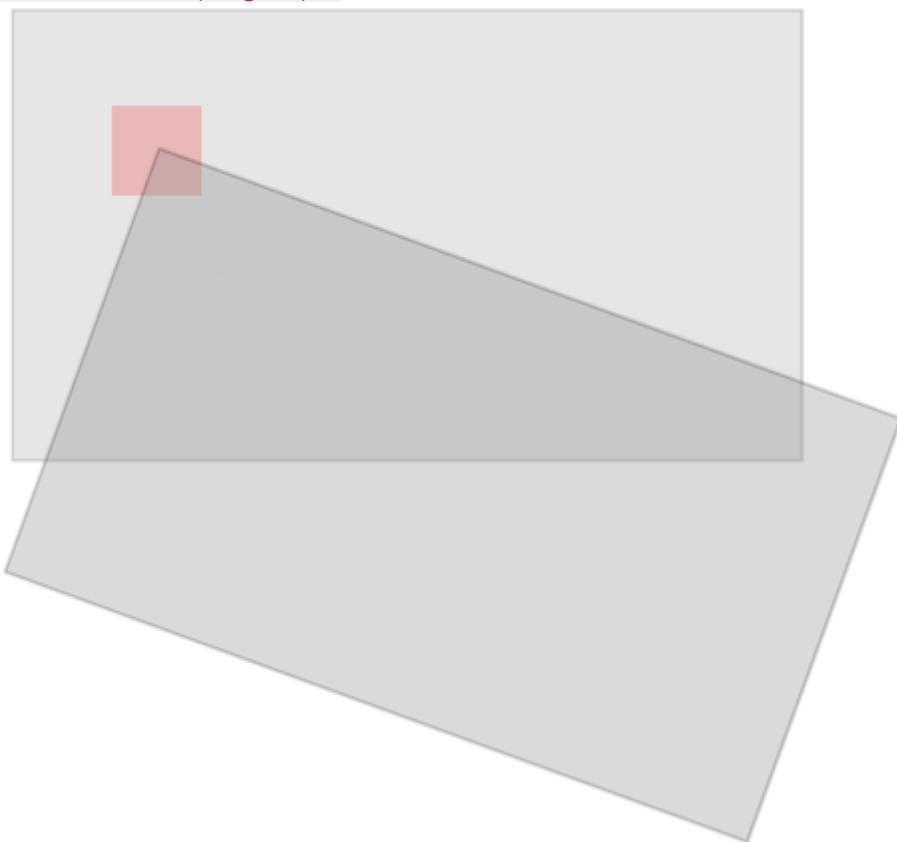
Then we move the entire canvas to the center of the specific component, using the translate method:

```
ctx.translate(x, y);
```



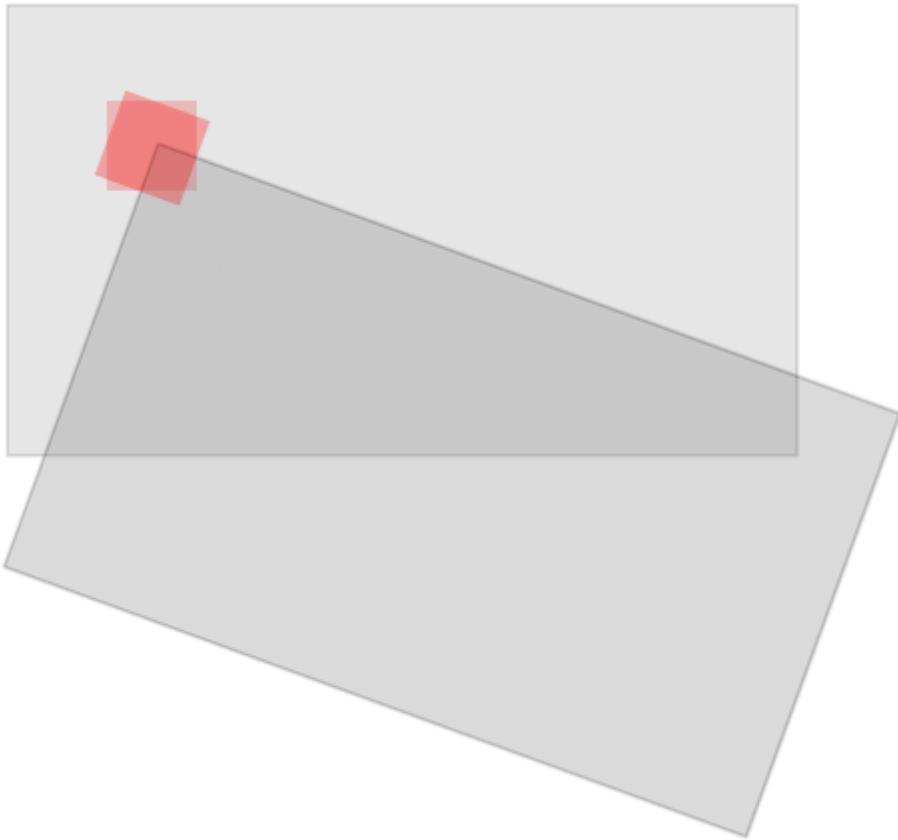
Then we perform the wanted rotation using the `rotate()` method:

```
ctx.rotate(angle);
```



Now we are ready to draw the component onto the canvas, but now we will draw it with its center position at position 0,0 on the translated (and rotated) canvas:

```
ctx.fillRect(width / -2, height / -2, width, height);
```



When we are finished, we must restore the context object back to its saved position, using the `restore` method:

```
ctx.restore();
```

The component is the only thing that is rotated:



The Component Constructor

The `component` constructor has a new property called `angle`, which is radian number that represents the angle of the component.

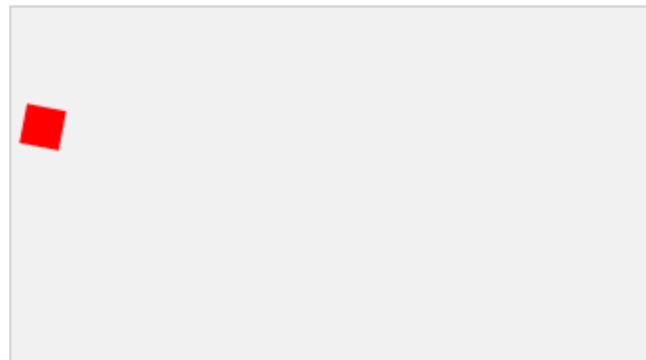
The `update` method of the `component` constructor is where we draw the component, and here you can see the changes that will allow the component to rotate:

Example

```
function component(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.angle = 0;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.save();
        ctx.translate(this.x, this.y);
        ctx.rotate(this.angle);
        ctx.fillStyle = color;
        ctx.fillRect(this.width / -2, this.height / -2, this.width,
this.height);
        ctx.restore();
    }
}
function updateGameArea() {
    myGameArea.clear();
    myGamePiece.angle += 1 * Math.PI / 180;
    myGamePiece.update();
}
```

Game Movement

With the new way of drawing components, explained in the Game Rotation chapter, the movements are more flexible.



Play again

How to Move Objects?

Add a `speed` property to the `component` constructor, which represents the current speed of the component.

Also make some changes in the `newPos()` method, to calculate the position of the component, based on `speed` and `angle`.

By default, the components are facing up, and by setting the speed property to 1, the component will start moving forward.

Example

```
function component(width, height, color, x, y) {
    this.gamearea = gamearea;
    this.width = width;
    this.height = height;
    this.angle = 0;
    this.speed = 1;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.save();
        ctx.translate(this.x, this.y);
        ctx.rotate(this.angle);
        ctx.fillStyle = color;
        ctx.fillRect(this.width / -2, this.height / -2, this.width,
this.height);
        ctx.restore();
    }
    this.newPos = function() {
        this.x += this.speed * Math.sin(this.angle);
        this.y -= this.speed * Math.cos(this.angle);
    }
}
```

Making Turns

We also want to be able to make left and right turns. Make a new property called `moveAngle`, which indicates the current moving value, or rotation angle. In the `newPos()` method calculate the `angle` based on the `moveAngle` property:

Example

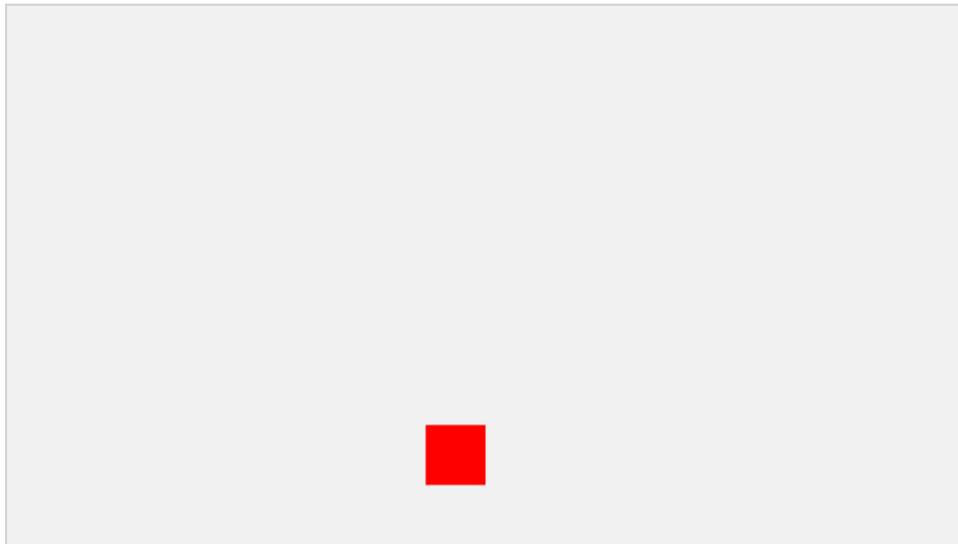
Set the `moveangle` property to 1, and see what happens:

```
function component(width, height, color, x, y) {
    this.width = width;
    this.height = height;
    this.angle = 0;
    this.moveAngle = 1;
    this.speed = 1;
    this.x = x;
    this.y = y;
    this.update = function() {
        ctx = myGameArea.context;
        ctx.save();
        ctx.translate(this.x, this.y);
        ctx.rotate(this.angle);
        ctx.fillStyle = color;
        ctx.fillRect(this.width / -2, this.height / -2, this.width,
this.height);
        ctx.restore();
    }
    this.newPos = function() {
        this.angle += this.moveAngle * Math.PI / 180;
        this.x += this.speed * Math.sin(this.angle);
        this.y -= this.speed * Math.cos(this.angle);
    }
}
```

Use the Keyboard

How does the red square move when using the keyboard? Instead of moving up and down, and from side to side, the red square moves forward when you use the "up" arrow, and turns left and right when pressing the left and right arrows.

Example



Make sure the gamearea has focus, and use the arrow keys to move the red square around.

Canvas Clock

In these chapters we will build an analog clock using HTML canvas.

Part I - Create the Canvas

The clock needs an HTML container. Create an HTML canvas:

HTML code:

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

<canvas id="canvas" width="400" height="400" style="background-color:#333">
</canvas>

<script>
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var radius = canvas.height / 2;
ctx.translate(radius, radius);
radius = radius * 0.90
drawClock();

function drawClock() {
    ctx.arc(0, 0, radius, 0 , 2*Math.PI);
    ctx.fillStyle = "white";
    ctx.fill();
}
</script>

</body>
</html>
```

Code Explained

Add an HTML <canvas> element to your page:

```
<canvas id="canvas" width="400" height="400" style="background-color:#333">
</canvas>
```

Create a canvas object (var canvas) from the HTML canvas element:

```
var canvas = document.getElementById("canvas");
```

Create a 2d drawing object (var ctx) for the canvas object:

```
var ctx = canvas.getContext("2d");
```

Calculate the clock radius, using the height of the canvas:

```
var radius = canvas.height / 2;
```

Using the canvas height to calculate the clock radius, makes the clock work for all canvas sizes.

Remap the (0,0) position (of the drawing object) to the center of the canvas:

```
ctx.translate(radius, radius);
```

Reduce the clock radius (to 90%) to draw the clock well inside the canvas:

```
radius = radius * 0.90;
```

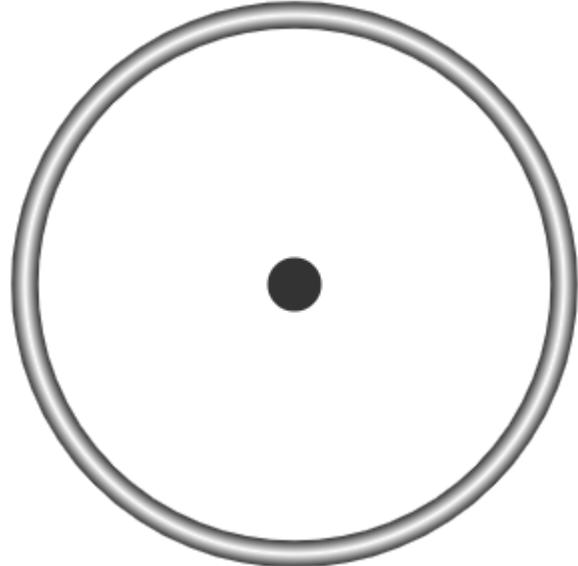
Create a function to draw the clock:

```
function drawClock() {  
    ctx.arc(0, 0, radius, 0 , 2*Math.PI);  
    ctx.fillStyle = "white";  
    ctx.fill();  
}
```

Canvas Clock Face

Part II - Draw a Clock Face

The clock needs a clock face. Create a JavaScript function to draw a clock face:



JavaScript:

```
function drawClock() {
    drawFace(ctx, radius);
}

function drawFace(ctx, radius) {
    var grad;

    ctx.beginPath();
    ctx.arc(0, 0, radius, 0, 2*Math.PI);
    ctx.fillStyle = 'white';
    ctx.fill();

    grad = ctx.createRadialGradient(0,0,radius*0.95, 0,0,radius*1.05);
    grad.addColorStop(0, '#333');
    grad.addColorStop(0.5, 'white');
    grad.addColorStop(1, '#333');
    ctx.strokeStyle = grad;
    ctx.lineWidth = radius*0.1;
    ctx.stroke();

    ctx.beginPath();
    ctx.arc(0, 0, radius*0.1, 0, 2*Math.PI);
    ctx.fillStyle = '#333';
    ctx.fill();
}
```

Code Explained

Create a `drawFace()` function for drawing the clock face:

```
function drawClock() {
    drawFace(ctx, radius);
}

function drawFace(ctx, radius) {
```

Draw the white circle:

```
ctx.beginPath();
ctx.arc(0, 0, radius, 0, 2*Math.PI);
ctx.fillStyle = 'white';
ctx.fill();
```

Create a radial gradient (95% and 105% of original clock radius):

```
grad = ctx.createRadialGradient(0,0,radius*0.95, 0,0,radius*1.05);
```

Create 3 color stops, corresponding with the inner, middle, and outer edge of the arc:

```
grad.addColorStop(0, '#333');
grad.addColorStop(0.5, 'white');
grad.addColorStop(1, '#333');
```

The color stops create a 3D effect.

Define the gradient as the stroke style of the drawing object:

```
ctx.strokeStyle = grad;
```

Define the line width of the drawing object (10% of radius):

```
ctx.lineWidth = radius * 0.1;
```

Draw the circle:

```
ctx.stroke();
```

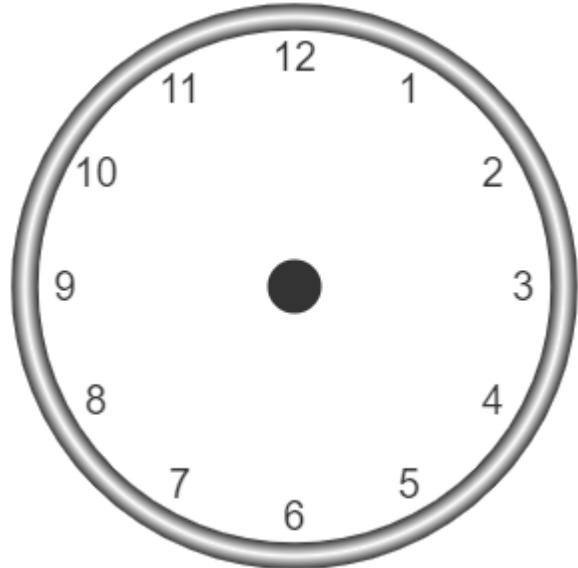
Draw the clock center:

```
ctx.beginPath();
ctx.arc(0, 0, radius*0.1, 0, 2*Math.PI);
ctx.fillStyle = '#333';
ctx.fill();
```

Canvas Clock Numbers

Part III - Draw Clock Numbers

The clock needs numbers. Create a JavaScript function to draw clock numbers:



JavaScript:

```
function drawClock() {
    drawFace(ctx, radius);
    drawNumbers(ctx, radius);
}

function drawNumbers(ctx, radius) {
    var ang;
    var num;
    ctx.font = radius*0.15 + "px arial";
    ctx.textBaseline="middle";
    ctx.textAlign="center";
    for(num= 1; num < 13; num++){
        ang = num * Math.PI / 6;
        ctx.rotate(ang);
        ctx.translate(0, -radius*0.85);
        ctx.rotate(-ang);
        ctx.fillText(num.toString(), 0, 0);
        ctx.rotate(ang);
        ctx.translate(0, radius*0.85);
        ctx.rotate(-ang);
    }
}
```

Example Explained

Set the font size (of the drawing object) to 15% of the radius:

```
ctx.font = radius*0.15 + "px arial";
```

Set the text alignment to the middle and the center of the print position:

```
ctx.textBaseline="middle";
ctx.textAlign="center";
```

Calculate the print position (for 12 numbers) to 85% of the radius, rotated ($\text{PI}/6$) for each number:

```
for(num= 1; num < 13; num++) {
    ang = num * Math.PI / 6;
    ctx.rotate(ang);
    ctx.translate(0, -radius*0.85);
    ctx.rotate(-ang);
    ctx.fillText(num.toString(), 0, 0);
    ctx.rotate(ang);
    ctx.translate(0, radius*0.85);
    ctx.rotate(-ang);
}
```

Canvas Clock Hands

Part IV - Draw Clock Hands

The clock needs hands. Create a JavaScript function to draw clock hands:



JavaScript:

```
function drawClock() {
    drawFace(ctx, radius);
    drawNumbers(ctx, radius);
    drawTime(ctx, radius);
}

function drawTime(ctx, radius){
    var now = new Date();
    var hour = now.getHours();
    var minute = now.getMinutes();
    var second = now.getSeconds();
    //hour
    hour=hour%12;
    hour=(hour*Math.PI/6)+(minute*Math.PI/(6*60))+(
(second*Math.PI/(360*60)));
    drawHand(ctx, hour, radius*0.5, radius*0.07);
    //minute
    minute=(minute*Math.PI/30)+(second*Math.PI/(30*60));
    drawHand(ctx, minute, radius*0.8, radius*0.07);
    // second
    second=(second*Math.PI/30);
    drawHand(ctx, second, radius*0.9, radius*0.02);
}

function drawHand(ctx, pos, length, width) {
    ctx.beginPath();
    ctx.lineWidth = width;
    ctx.lineCap = "round";
    ctx.moveTo(0,0);
    ctx.rotate(pos);
    ctx.lineTo(0, -length);
    ctx.stroke();
    ctx.rotate(-pos);
}
```

Example Explained

Use Date to get hour, minute, second:

```
var now = new Date();
var hour = now.getHours();
var minute = now.getMinutes();
var second = now.getSeconds();
```

Calculate the angle of the hour hand, and draw it a length (50% of radius), and a width (7% of radius):

```
hour=hour%12;
hour=(hour*Math.PI/6)+(minute*Math.PI/(6*60))+(second*Math.PI/(360*60));
drawHand(ctx, hour, radius*0.5, radius*0.07);
```

Use the same technique for minutes and seconds.

The drawHand() routine does not need an explanation. It just draws a line with a given length and width.

Canvas Clock Start

In these chapters we build an analog clock using HTML Canvas.

Part V - Start the Clock

To start the clock, call the drawClock function at intervals:



JavaScript:

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var radius = canvas.height / 2;
ctx.translate(radius, radius);
radius = radius * 0.90
//drawClock();
setInterval(drawClock, 1000);
```

Example Explained

The only thing you have to do (to start the clock) is to call the drawClock function at intervals.
Substitute:

```
drawClock();
```

With:

```
setInterval(drawClock, 1000);
```

The interval is in milliseconds. drawClock will be called for each 1000 milliseconds.