HTML Tutorial

Top of Form

Bottom of Form

**HTML** stands for **Hyper Text Markup Language**, which is the most widely used language on Web to develop web pages. **HTML** was created by Berners-Lee in late 1991 but "HTML 2.0" was the first standard HTML specification which was published in 1995. HTML 4.01 was a major version of HTML and it was published in late 1999. Though HTML 4.01 version is widely used but currently we are having HTML-5 version which is an extension to HTML 4.01, and this version was published in 2012.

Why to Learn HTML?

Originally, **HTML** was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers. Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

**HTML** is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning HTML:

* **Create Web site** - You can create a website or customize an existing web template if you know HTML well.
* **Become a web designer** - If you want to start a carrer as a professional web designer, HTML and CSS designing is a must skill.
* **Understand web** - If you want to optimize your website, to boost its speed and performance, it is good to know HTML to yield best results.
* **Learn other languages** - Once you understands the basic of HTML then other related technologies like javascript, php, or angular are become easier to understand.

Hello World using HTML.

Just to give you a little excitement about HTML, I'm going to give you a small conventional **HTML Hello World** program, You can try it using Demo link.

[Live Demo](http://tpcg.io/r2QSCf)

<!DOCTYPE html>

<html>

<head>

<title>This is document title</title>

</head>

<body>

<h1>This is a heading</h1>

<p>Hello World!</p>

</body>

</html>

Applications of HTML

As mentioned before, HTML is one of the most widely used language over the web. I'm going to list few of them here:

* **Web pages development** - HTML is used to create pages which are rendered over the web. Almost every page of web is having html tags in it to render its details in browser.
* **Internet Navigation** - HTML provides tags which are used to navigate from one page to another and is heavily used in internet navigation.
* **Responsive UI** - HTML pages now-a-days works well on all platform, mobile, tabs, desktop or laptops owing to responsive design strategy.
* **Offline support** HTML pages once loaded can be made available offline on the machine without any need of internet.
* **Game development**- HTML5 has native support for rich experience and is now useful in gaming developent arena as well.

Audience

This **HTML tutorial** is designed for the aspiring Web Designers and Developers with a need to understand the HTML in enough detail along with its simple overview, and practical examples. This tutorial will give you enough ingredients to start with HTML from where you can take yourself at higher level of expertise.

Prerequisites

Before proceeding with this **tutorial** you should have a basic working knowledge with Windows or Linux operating system, additionally you must be familiar with −

* Experience with any text editor like notepad, notepad++, or Edit plus etc.
* How to create directories and files on your computer.
* How to navigate through different directories.
* How to type content in a file and save them on a computer.
* Understanding about images in different formats like JPEG, PNG format.

HTML - Overview

Advertisements

Ad by Valueimpression

[Previous Page](https://www.tutorialspoint.com/html/index.htm)

[Next Page](https://www.tutorialspoint.com/html/html_basic_tags.htm)

HTML stands for **H**yper**t**ext **M**arkup **L**anguage, and it is the most widely used language to write Web Pages.

* **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a webpage is called Hypertext.
* As its name suggests, HTML is a **Markup Language** which means you use HTML to simply "mark-up" a text document with tags that tell a Web browser how to structure it to display.

Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers.

Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

Basic HTML Document

In its simplest form, following is an example of an HTML document −

[Live Demo](http://tpcg.io/tryL9g)

<!DOCTYPE html>

<html>

<head>

<title>This is document title</title>

</head>

<body>

<h1>This is a heading</h1>

<p>Document content goes here.....</p>

</body>

</html>

HTML Tags

As told earlier, HTML is a markup language and makes use of various tags to format the content. These tags are enclosed within angle braces **<Tag Name>**. Except few tags, most of the tags have their corresponding closing tags. For example, **<html>** has its closing tag **</html>** and **<body>** tag has its closing tag **</body>** tag etc.

Above example of HTML document uses the following tags −

|  |  |
| --- | --- |
| **Sr.No** | **Tag & Description** |
| 1 | **<!DOCTYPE...>**  This tag defines the document type and HTML version. |
| 2 | **<html>**  This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags. |
| 3 | **<head>**  This tag represents the document's header which can keep other HTML tags like <title>, <link> etc. |
| 4 | **<title>**  The <title> tag is used inside the <head> tag to mention the document title. |
| 5 | **<body>**  This tag represents the document's body which keeps other HTML tags like <h1>, <div>, <p> etc. |
| 6 | **<h1>**  This tag represents the heading. |
| 7 | **<p>**  This tag represents a paragraph. |

To learn HTML, you will need to study various tags and understand how they behave, while formatting a textual document. Learning HTML is simple as users have to learn the usage of different tags in order to format the text or images to make a beautiful webpage.

World Wide Web Consortium (W3C) recommends to use lowercase tags starting from HTML 4.

HTML Document Structure

A typical HTML document will have the following structure −

<html>

<head>

Document header related tags

</head>

<body>

Document body related tags

</body>

</html>

We will study all the header and body tags in subsequent chapters, but for now let's see what is document declaration tag.

The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration tag is used by the web browser to understand the version of the HTML used in the document. Current version of HTML is 5 and it makes use of the following declaration −

<!DOCTYPE html>

There are many other declaration types which can be used in HTML document depending on what version of HTML is being used. We will see more details on this while discussing <!DOCTYPE...> tag along with other HTML tags.

# HTML - Basic Tags

Advertisements

Ad by Valueimpression

[Previous Page](https://www.tutorialspoint.com/html/html_overview.htm)

[Next Page](https://www.tutorialspoint.com/html/html_elements.htm)

## Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>, <h2>, <h3>, <h4>, <h5>,** and **<h6>**. While displaying any heading, browser adds one line before and one line after that heading.

### Example

[Live Demo](http://tpcg.io/Oevfe8)

<!DOCTYPE html>

<html>

<head>

<title>Heading Example</title>

</head>

<body>

<h1>This is heading 1</h1>

<h2>This is heading 2</h2>

<h3>This is heading 3</h3>

<h4>This is heading 4</h4>

<h5>This is heading 5</h5>

<h6>This is heading 6</h6>

</body>

</html>

This will produce the following result −

## Paragraph Tag

The **<p>** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening <p> and a closing </p> tag as shown below in the example −

### Example

[Live Demo](http://tpcg.io/Uo5jZe)

<!DOCTYPE html>

<html>

<head>

<title>Paragraph Example</title>

</head>

<body>

<p>Here is a first paragraph of text.</p>

<p>Here is a second paragraph of text.</p>

<p>Here is a third paragraph of text.</p>

</body>

</html>

This will produce the following result −

## Line Break Tag

Whenever you use the **<br />** element, anything following it starts from the next line. This tag is an example of an **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The <br /> tag has a space between the characters **br** and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use <br> it is not valid in XHTML.

### Example

[Live Demo](http://tpcg.io/aWg5PG)

<!DOCTYPE html>

<html>

<head>

<title>Line Break Example</title>

</head>

<body>

<p>Hello<br />

You delivered your assignment ontime.<br />

Thanks<br />

Mahnaz</p>

</body>

</html>

This will produce the following result −

## Centering Content

You can use **<center>** tag to put any content in the center of the page or any table cell.

### Example

[Live Demo](http://tpcg.io/mVrRPS)

<!DOCTYPE html>

<html>

<head>

<title>Centring Content Example</title>

</head>

<body>

<p>This text is not in the center.</p>

<center>

<p>This text is in the center.</p>

</center>

</body>

</html>

This will produce following result −

## Horizontal Lines

Horizontal lines are used to visually break-up sections of a document. The **<hr>** tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

For example, you may want to give a line between two paragraphs as in the given example below −

### Example

[Live Demo](http://tpcg.io/4iAnXc)

<!DOCTYPE html>

<html>

<head>

<title>Horizontal Line Example</title>

</head>

<body>

<p>This is paragraph one and should be on top</p>

<hr />

<p>This is paragraph two and should be at bottom</p>

</body>

</html>

This will produce the following result −

Again **<hr />** tag is an example of the **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The **<hr />** element has a space between the characters **hr** and the forward slash. If you omit this space, older browsers will have trouble rendering the horizontal line, while if you miss the forward slash character and just use **<hr>** it is not valid in XHTML

## Preserve Formatting

Sometimes, you want your text to follow the exact format of how it is written in the HTML document. In these cases, you can use the preformatted tag **<pre>**.

Any text between the opening **<pre>** tag and the closing **</pre>** tag will preserve the formatting of the source document.

### Example

[Live Demo](http://tpcg.io/e3k3sG)

<!DOCTYPE html>

<html>

<head>

<title>Preserve Formatting Example</title>

</head>

<body>

<pre>

function testFunction( strText ){

alert (strText)

}

</pre>

</body>

</html>

This will produce the following result −

Try using the same code without keeping it inside **<pre>...</pre>** tags

## Nonbreaking Spaces

Suppose you want to use the phrase "12 Angry Men." Here, you would not want a browser to split the "12, Angry" and "Men" across two lines −

An example of this technique appears in the movie "12 Angry Men."

In cases, where you do not want the client browser to break text, you should use a nonbreaking space entity **&nbsp;** instead of a normal space. For example, when coding the "12 Angry Men" in a paragraph, you should use something similar to the following code −

### Example

[Live Demo](http://tpcg.io/OhhfWR)

<!DOCTYPE html>

<html>

<head>

<title>Nonbreaking Spaces Example</title>

</head>

<body>

<p>An example of this technique appears in the movie "12&nbsp;Angry&nbsp;Men."</p>

</body>

</html>

This will produce the following result −

# HTML - Elements

An **HTML element** is defined by a starting tag. If the element contains other content, it ends with a closing tag, where the element name is preceded by a forward slash as shown below with few tags −

|  |  |  |
| --- | --- | --- |
| **Start Tag** | **Content** | **End Tag** |
| <p> | This is paragraph content. | </p> |
| <h1> | This is heading content. | </h1> |
| <div> | This is division content. | </div> |
| <br /> |  |  |

So here **<p>....</p>** is an HTML element, **<h1>...</h1>** is another HTML element. There are some HTML elements which don't need to be closed, such as **<img.../>**, **<hr />** and **<br />** elements. These are known as **void elements**.

HTML documents consists of a tree of these elements and they specify how HTML documents should be built, and what kind of content should be placed in what part of an HTML document.

## HTML Tag vs. Element

An HTML element is defined by a *starting tag*. If the element contains other content, it ends with a *closing tag*.

For example, **<p>** is starting tag of a paragraph and **</p>** is closing tag of the same paragraph but **<p>This is paragraph</p>** is a paragraph element.

## Nested HTML Elements

It is very much allowed to keep one HTML element inside another HTML element −

### Example

[Live Demo](http://tpcg.io/RWo6kx)

<!DOCTYPE html>

<html>

<head>

<title>Nested Elements Example</title>

</head>

<body>

<h1>This is <i>italic</i> heading</h1>

<p>This is <u>underlined</u> paragraph</p>

</body>

</html>

This will display the following result −

# HTML - Attributes

We have seen few HTML tags and their usage like heading tags **<h1>, <h2>,** paragraph tag **<p>** and other tags. We used them so far in their simplest form, but most of the HTML tags can also have attributes, which are extra bits of information.

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts − a **name** and a **value**

* The **name** is the property you want to set. For example, the paragraph **<p>** element in the example carries an attribute whose name is **align**, which you can use to indicate the alignment of paragraph on the page.
* The **value** is what you want the value of the property to be set and always put within quotations. The below example shows three possible values of align attribute: **left, center** and **right**.

Attribute names and attribute values are case-insensitive. However, the World Wide Web Consortium (W3C) recommends lowercase attributes/attribute values in their HTML 4 recommendation.

### Example

[Live Demo](http://tpcg.io/rJQ2Po)

<!DOCTYPE html>

<html>

<head>

<title>Align Attribute Example</title>

</head>

<body>

<p align = "left">This is left aligned</p>

<p align = "center">This is center aligned</p>

<p align = "right">This is right aligned</p>

</body>

</html>

This will display the following result −

## Core Attributes

The four core attributes that can be used on the majority of HTML elements (although not all) are −

* Id
* Title
* Class
* Style

### The Id Attribute

The **id** attribute of an HTML tag can be used to uniquely identify any element within an HTML page. There are two primary reasons that you might want to use an id attribute on an element −

* If an element carries an id attribute as a unique identifier, it is possible to identify just that element and its content.
* If you have two elements of the same name within a Web page (or style sheet), you can use the id attribute to distinguish between elements that have the same name.

We will discuss style sheet in separate tutorial. For now, let's use the id attribute to distinguish between two paragraph elements as shown below.

**Example**

<p id = "html">This para explains what is HTML</p>

<p id = "css">This para explains what is Cascading Style Sheet</p>

### The title Attribute

The **title** attribute gives a suggested title for the element. They syntax for the **title** attribute is similar as explained for **id** attribute −

The behavior of this attribute will depend upon the element that carries it, although it is often displayed as a tooltip when cursor comes over the element or while the element is loading.

**Example**

[Live Demo](http://tpcg.io/Xq2ok1)

<!DOCTYPE html>

<html>

<head>

<title>The title Attribute Example</title>

</head>

<body>

<h3 title = "Hello HTML!">Titled Heading Tag Example</h3>

</body>

</html>

This will produce the following result −

Now try to bring your cursor over "Titled Heading Tag Example" and you will see that whatever title you used in your code is coming out as a tooltip of the cursor.

### The class Attribute

The **class** attribute is used to associate an element with a style sheet, and specifies the class of element. You will learn more about the use of the class attribute when you will learn Cascading Style Sheet (CSS). So for now you can avoid it.

The value of the attribute may also be a space-separated list of class names. For example −

class = "className1 className2 className3"

### The style Attribute

The style attribute allows you to specify Cascading Style Sheet (CSS) rules within the element.

[Live Demo](http://tpcg.io/FreFRI)

<!DOCTYPE html>

<html>

<head>

<title>The style Attribute</title>

</head>

<body>

<p style = "font-family:arial; color:#FF0000;">Some text...</p>

</body>

</html>

This will produce the following result −

At this point of time, we are not learning CSS, so just let's proceed without bothering much about CSS. Here, you need to understand what are HTML attributes and how they can be used while formatting content.

## Internationalization Attributes

There are three internationalization attributes, which are available for most (although not all) XHTML elements.

* dir
* lang
* xml:lang

### The dir Attribute

The **dir** attribute allows you to indicate to the browser about the direction in which the text should flow. The dir attribute can take one of two values, as you can see in the table that follows −

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| ltr | Left to right (the default value) |
| rtl | Right to left (for languages such as Hebrew or Arabic that are read right to left) |

**Example**

[Live Demo](http://tpcg.io/a9JBAz)

<!DOCTYPE html>

<html dir = "rtl">

<head>

<title>Display Directions</title>

</head>

<body>

This is how IE 5 renders right-to-left directed text.

</body>

</html>

This will produce the following result −

When *dir* attribute is used within the <html> tag, it determines how text will be presented within the entire document. When used within another tag, it controls the text's direction for just the content of that tag.

### The lang Attribute

The **lang** attribute allows you to indicate the main language used in a document, but this attribute was kept in HTML only for backwards compatibility with earlier versions of HTML. This attribute has been replaced by the **xml:lang** attribute in new XHTML documents.

The values of the *lang* attribute are ISO-639 standard two-character language codes. Check [**HTML Language Codes: ISO 639**](https://www.tutorialspoint.com/html/language_iso_codes.htm) for a complete list of language codes.

**Example**

[Live Demo](http://tpcg.io/Hf0XPh)

<!DOCTYPE html>

<html lang = "en">

<head>

<title>English Language Page</title>

</head>

<body>

This page is using English Language

</body>

</html>

This will produce the following result −

## The xml:lang Attribute

The *xml:lang* attribute is the XHTML replacement for the *lang* attribute. The value of the *xml:lang* attribute should be an ISO-639 country code as mentioned in previous section.

### Generic Attributes

Here's a table of some other attributes that are readily usable with many of the HTML tags.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Options** | **Function** |
| align | right, left, center | Horizontally aligns tags |
| valign | top, middle, bottom | Vertically aligns tags within an HTML element. |
| bgcolor | numeric, hexidecimal, RGB values | Places a background color behind an element |
| background | URL | Places a background image behind an element |
| id | User Defined | Names an element for use with Cascading Style Sheets. |
| class | User Defined | Classifies an element for use with Cascading Style Sheets. |
| width | Numeric Value | Specifies the width of tables, images, or table cells. |
| height | Numeric Value | Specifies the height of tables, images, or table cells. |
| title | User Defined | "Pop-up" title of the elements. |

# HTML - Formatting

If you use a word processor, you must be familiar with the ability to make text bold, italicized, or underlined; these are just three of the ten options available to indicate how text can appear in HTML and XHTML.

## Bold Text

Anything that appears within **<b>...</b>** element, is displayed in bold as shown below −

### Example

[Live Demo](http://tpcg.io/5KMKkW)

<!DOCTYPE html>

<html>

<head>

<title>Bold Text Example</title>

</head>

<body>

<p>The following word uses a <b>bold</b> typeface.</p>

</body>

</html>

This will produce the following result −

## Italic Text

Anything that appears within **<i>...</i>** element is displayed in italicized as shown below −

### Example

[Live Demo](http://tpcg.io/OJgqcS)

<!DOCTYPE html>

<html>

<head>

<title>Italic Text Example</title>

</head>

<body>

<p>The following word uses an <i>italicized</i> typeface.</p>

</body>

</html>

This will produce the following result −

## Underlined Text

Anything that appears within **<u>...</u>** element, is displayed with underline as shown below −

### Example

[Live Demo](http://tpcg.io/5Y75MX)

<!DOCTYPE html>

<html>

<head>

<title>Underlined Text Example</title>

</head>

<body>

<p>The following word uses an <u>underlined</u> typeface.</p>

</body>

</html>

This will produce the following result −

## Strike Text

Anything that appears within **<strike>...</strike>** element is displayed with strikethrough, which is a thin line through the text as shown below −

### Example

[Live Demo](http://tpcg.io/HIkU7n)

<!DOCTYPE html>

<html>

<head>

<title>Strike Text Example</title>

</head>

<body>

<p>The following word uses a <strike>strikethrough</strike> typeface.</p>

</body>

</html>

This will produce the following result −

## Monospaced Font

The content of a **<tt>...</tt>** element is written in monospaced font. Most of the fonts are known as variable-width fonts because different letters are of different widths (for example, the letter 'm' is wider than the letter 'i'). In a monospaced font, however, each letter has the same width.

### Example

[Live Demo](http://tpcg.io/7XdwHl)

<!DOCTYPE html>

<html>

<head>

<title>Monospaced Font Example</title>

</head>

<body>

<p>The following word uses a <tt>monospaced</tt> typeface.</p>

</body>

</html>

This will produce the following result −

## Superscript Text

The content of a **<sup>...</sup>** element is written in superscript; the font size used is the same size as the characters surrounding it but is displayed half a character's height above the other characters.

### Example

[Live Demo](http://tpcg.io/ASSDpm)

<!DOCTYPE html>

<html>

<head>

<title>Superscript Text Example</title>

</head>

<body>

<p>The following word uses a <sup>superscript</sup> typeface.</p>

</body>

</html>

This will produce the following result −

## Subscript Text

The content of a **<sub>...</sub>** element is written in subscript; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.

### Example

[Live Demo](http://tpcg.io/FS24k4)

<!DOCTYPE html>

<html>

<head>

<title>Subscript Text Example</title>

</head>

<body>

<p>The following word uses a <sub>subscript</sub> typeface.</p>

</body>

</html>

This will produce the following result −

## Inserted Text

Anything that appears within **<ins>...</ins>** element is displayed as inserted text.

### Example

[Live Demo](http://tpcg.io/8WK66t)

<!DOCTYPE html>

<html>

<head>

<title>Inserted Text Example</title>

</head>

<body>

<p>I want to drink <del>cola</del> <ins>wine</ins></p>

</body>

</html>

This will produce the following result −

## Deleted Text

Anything that appears within **<del>...</del>** element, is displayed as deleted text.

### Example

[Live Demo](http://tpcg.io/HAK4s7)

<!DOCTYPE html>

<html>

<head>

<title>Deleted Text Example</title>

</head>

<body>

<p>I want to drink <del>cola</del> <ins>wine</ins></p>

</body>

</html>

This will produce the following result −

## Larger Text

The content of the **<big>...</big>** element is displayed one font size larger than the rest of the text surrounding it as shown below −

### Example

[Live Demo](http://tpcg.io/XGxTMW)

<!DOCTYPE html>

<html>

<head>

<title>Larger Text Example</title>

</head>

<body>

<p>The following word uses a <big>big</big> typeface.</p>

</body>

</html>

This will produce the following result −

## Smaller Text

The content of the **<small>...</small>** element is displayed one font size smaller than the rest of the text surrounding it as shown below −

### Example

[Live Demo](http://tpcg.io/BtVDGJ)

<!DOCTYPE html>

<html>

<head>

<title>Smaller Text Example</title>

</head>

<body>

<p>The following word uses a <small>small</small> typeface.</p>

</body>

</html>

This will produce the following result −

## Grouping Content

The **<div>** and **<span>** elements allow you to group together several elements to create sections or subsections of a page.

For example, you might want to put all of the footnotes on a page within a <div> element to indicate that all of the elements within that <div> element relate to the footnotes. You might then attach a style to this <div> element so that they appear using a special set of style rules.

### Example

[Live Demo](http://tpcg.io/tqWkee)

<!DOCTYPE html>

<html>

<head>

<title>Div Tag Example</title>

</head>

<body>

<div id = "menu" align = "middle" >

<a href = "/index.htm">HOME</a> |

<a href = "/about/contact\_us.htm">CONTACT</a> |

<a href = "/about/index.htm">ABOUT</a>

</div>

<div id = "content" align = "left" bgcolor = "white">

<h5>Content Articles</h5>

<p>Actual content goes here.....</p>

</div>

</body>

</html>

This will produce the following result −

The <span> element, on the other hand, can be used to group inline elements only. So, if you have a part of a sentence or paragraph which you want to group together, you could use the <span> element as follows.

### Example

[Live Demo](http://tpcg.io/R5ONzp)

<!DOCTYPE html>

<html>

<head>

<title>Span Tag Example</title>

</head>

<body>

<p>This is the example of <span style = "color:green">span tag</span>

and the <span style = "color:red">div tag</span> alongwith CSS</p>

</body>

</html>

This will produce the following result −

# HTML - Phrase Tags

The phrase tags have been desicolgned for specific purposes, though they are displayed in a similar way as other basic tags like **<b>, <i>, <pre>**, and **<tt>**, you have seen in previous chapter. This chapter will take you through all the important phrase tags, so let's start seeing them one by one.

## Emphasized Text

Anything that appears within **<em>...</em>** element is displayed as emphasized text.

### Example

[Live Demo](http://tpcg.io/kT5EwY)

<!DOCTYPE html>

<html>

<head>

<title>Emphasized Text Example</title>

</head>

<body>

<p>The following word uses an <em>emphasized</em> typeface.</p>

</body>

</html>

This will produce the following result −

## Marked Text

Anything that appears with-in **<mark>...</mark>** element, is displayed as marked with yellow ink.

### Example

[Live Demo](http://tpcg.io/nBGxW3)

<!DOCTYPE html>

<html>

<head>

<title>Marked Text Example</title>

</head>

<body>

<p>The following word has been <mark>marked</mark> with yellow</p>

</body>

</html>

This will produce the following result −

## Strong Text

Anything that appears within **<strong>...</strong>** element is displayed as important text.

### Example

[Live Demo](http://tpcg.io/VHJbmy)

<!DOCTYPE html>

<html>

<head>

<title>Strong Text Example</title>

</head>

<body>

<p>The following word uses a <strong>strong</strong> typeface.</p>

</body>

</html>

This will produce the following result −

## Text Abbreviation

You can abbreviate a text by putting it inside opening <abbr> and closing </abbr> tags. If present, the title attribute must contain this full description and nothing else.

### Example

[Live Demo](http://tpcg.io/htjF7E)

<!DOCTYPE html>

<html>

<head>

<title>Text Abbreviation</title>

</head>

<body>

<p>My best friend's name is <abbr title = "Abhishek">Abhy</abbr>.</p>

</body>

</html>

This will produce the following result −

## Acronym Element

The **<acronym>** element allows you to indicate that the text between <acronym> and </acronym> tags is an acronym.

At present, the major browsers do not change the appearance of the content of the <acronym> element.

### Example

[Live Demo](http://tpcg.io/lN65CX)

<!DOCTYPE html>

<html>

<head>

<title>Acronym Example</title>

</head>

<body>

<p>This chapter covers marking up text in <acronym>XHTML</acronym>.</p>

</body>

</html>

This will produce the following result −

## Text Direction

The **<bdo>...</bdo>** element stands for Bi-Directional Override and it is used to override the current text direction.

### Example

[Live Demo](http://tpcg.io/kEtHEG)

<!DOCTYPE html>

<html>

<head>

<title>Text Direction Example</title>

</head>

<body>

<p>This text will go left to right.</p>

<p><bdo dir = "rtl">This text will go right to left.</bdo></p>

</body>

</html>

This will produce the following result −

## Special Terms

The **<dfn>...</dfn>** element (or HTML Definition Element) allows you to specify that you are introducing a special term. It's usage is similar to italic words in the midst of a paragraph.

Typically, you would use the <dfn> element the first time you introduce a key term. Most recent browsers render the content of a <dfn> element in an italic font.

### Example

[Live Demo](http://tpcg.io/hrkKJl)

<!DOCTYPE html>

<html>

<head>

<title>Special Terms Example</title>

</head>

<body>

<p>The following word is a <dfn>special</dfn> term.</p>

</body>

</html>

This will produce the following result −

## Quoting Text

When you want to quote a passage from another source, you should put it in between **<blockquote>...</blockquote>** tags.

Text inside a <blockquote> element is usually indented from the left and right edges of the surrounding text, and sometimes uses an italicized font.

### Example

[Live Demo](http://tpcg.io/f2dsGJ)

<!DOCTYPE html>

<html>

<head>

<title>Blockquote Example</title>

</head>

<body>

<p>The following description of XHTML is taken from the W3C Web site:</p>

<blockquote>XHTML 1.0 is the W3C's first Recommendation for XHTML,following on

from earlier work on HTML 4.01, HTML 4.0, HTML 3.2 and HTML 2.0.</blockquote>

</body>

</html>

This will produce the following result −

## Short Quotations

The **<q>...</q>** element is used when you want to add a double quote within a sentence.

### Example

[Live Demo](http://tpcg.io/lOpgn9)

<!DOCTYPE html>

<html>

<head>

<title>Double Quote Example</title>

</head>

<body>

<p>Amit is in Spain, <q>I think I am wrong</q>.</p>

</body>

</html>

This will produce the following result −

## Text Citations

If you are quoting a text, you can indicate the source placing it between an opening **<cite>** tag and closing **</cite>** tag

As you would expect in a print publication, the content of the <cite> element is rendered in italicized text by default.

### Example

[Live Demo](http://tpcg.io/Efh84I)

<!DOCTYPE html>

<html>

<head>

<title>Citations Example</title>

</head>

<body>

<p>This HTML tutorial is derived from <cite>W3 Standard for HTML</cite>.</p>

</body>

</html>

This will produce the following result −

## Computer Code

Any programming code to appear on a Web page should be placed inside **<code>...</code>** tags. Usually the content of the <code> element is presented in a monospaced font, just like the code in most programming books.

### Example

[Live Demo](http://tpcg.io/YWhgcJ)

<!DOCTYPE html>

<html>

<head>

<title>Computer Code Example</title>

</head>

<body>

<p>Regular text. <code>This is code.</code> Regular text.</p>

</body>

</html>

This will produce the following result −

## Keyboard Text

When you are talking about computers, if you want to tell a reader to enter some text, you can use the **<kbd>...</kbd>** element to indicate what should be typed in, as in this example.

### Example

[Live Demo](http://tpcg.io/PjCgI1)

<!DOCTYPE html>

<html>

<head>

<title>Keyboard Text Example</title>

</head>

<body>

<p>Regular text. <kbd>This is inside kbd element</kbd> Regular text.</p>

</body>

</html>

This will produce the following result −

## Programming Variables

This element is usually used in conjunction with the **<pre>** and **<code>** elements to indicate that the content of that element is a variable.

### Example

[Live Demo](http://tpcg.io/p7iI1K)

<!DOCTYPE html>

<html>

<head>

<title>Variable Text Example</title>

</head>

<body>

<p><code>document.write("<var>user-name</var>")</code></p>

</body>

</html>

This will produce the following result −

## Program Output

The **<samp>...</samp>** element indicates sample output from a program, and script etc. Again, it is mainly used when documenting programming or coding concepts.

### Example

[Live Demo](http://tpcg.io/2V0rfa)

<!DOCTYPE html>

<html>

<head>

<title>Program Output Example</title>

</head>

<body>

<p>Result produced by the program is <samp>Hello World!</samp></p>

</body>

</html>

This will produce the following result −

## Address Text

The **<address>...</address>** element is used to contain any address.

### Example

[Live Demo](http://tpcg.io/L9qpv5)

<!DOCTYPE html>

<html>

<head>

<title>Address Example</title>

</head>

<body>

<address>388A, Road No 22, Jubilee Hills - Hyderabad</address>

</body>

</html>

This will produce the following result −

# HTML - Comments

Comment is a piece of code which is ignored by any web browser. It is a good practice to add comments into your HTML code, especially in complex documents, to indicate sections of a document, and any other notes to anyone looking at the code. Comments help you and others understand your code and increases code readability.

HTML comments are placed in between **<!-- ... -->** tags. So, any content placed with-in <!-- ... --> tags will be treated as comment and will be completely ignored by the browser.

### Example

[Live Demo](http://tpcg.io/YZPw02)

<!DOCTYPE html>

<html>

<head> <!-- Document Header Starts -->

<title>This is document title</title>

</head> <!-- Document Header Ends -->

<body>

<p>Document content goes here.....</p>

</body>

</html>

This will produce the following result without displaying the content given as a part of comments −

## Valid vs Invalid Comments

Comments do not nest which means a comment cannot be put inside another comment. Second the double-dash sequence "--" may not appear inside a comment except as part of the closing --> tag. You must also make sure that there are no spaces in the start-of comment string.

### Example

Here, the given comment is a valid comment and will be wiped off by the browser.

[Live Demo](http://tpcg.io/9RrhVS)

<!DOCTYPE html>

<html>

<head>

<title>Valid Comment Example</title>

</head>

<body>

<!-- This is valid comment -->

<p>Document content goes here.....</p>

</body>

</html>

This will produce the following result −

But, following line is not a valid comment and will be displayed by the browser. This is because there is a space between the left angle bracket and the exclamation mark.

[Live Demo](http://tpcg.io/WO8Y4y)

<!DOCTYPE html>

<html>

<head>

<title>Invalid Comment Example</title>

</head>

<body>

< !-- This is not a valid comment -->

<p>Document content goes here.....</p>

</body>

</html>

This will produce the following result −

## Multiline Comments

So far we have seen single line comments, but HTML supports multi-line comments as well.

You can comment multiple lines by the special beginning tag <!-- and ending tag --> placed before the first line and end of the last line as shown in the given example below.

### Example

[Live Demo](http://tpcg.io/UdjHu5)

<!DOCTYPE html>

<html>

<head>

<title>Multiline Comments</title>

</head>

<body>

<!--

This is a multiline comment and it can

span through as many as lines you like.

-->

<p>Document content goes here.....</p>

</body>

</html>

This will produce the following result −

## Conditional Comments

Conditional comments only work in Internet Explorer (IE) on Windows but they are ignored by other browsers. They are supported from Explorer 5 onwards, and you can use them to give conditional instructions to different versions of IE.

### Example

[Live Demo](http://tpcg.io/QAtpzr)

<!DOCTYPE html>

<html>

<head>

<title>Conditional Comments</title>

<!--[if IE 6]>

Special instructions for IE 6 here

<![endif]-->

</head>

<body>

<p>Document content goes here.....</p>

</body>

</html>

You will come across a situation where you will need to apply a different style sheet based on different versions of Internet Explorer, in such situation conditional comments will be helpful.

## Using Comment Tag

There are few browsers that support <comment> tag to comment a part of HTML code.

**Note** − The <comment> tag deprecated in HTML5. Do not use this element.

### Example

[Live Demo](http://tpcg.io/P2XIcz)

<!DOCTYPE html>

<html>

<head>

<title>Using Comment Tag</title>

</head>

<body>

<p>This is <comment>not</comment> Internet Explorer.</p>

</body>

</html>

If you are using IE, then it will produce following result −

But if you are not using IE, then it will produce following result −

## Commenting Script Code

Though you will learn JavaScript with HTML, in a separate tutorial, but here you must make a note that if you are using Java Script or VB Script in your HTML code then it is recommended to put that script code inside proper HTML comments so that old browsers can work properly.

### Example

[Live Demo](http://tpcg.io/BdDOlT)

<!DOCTYPE html>

<html>

<head>

<title>Commenting Script Code</title>

<script>

<!--

document.write("Hello World!")

//-->

</script>

</head>

<body>

<p>Hello , World!</p>

</body>

</html>

This will produce the following result −

## Commenting Style Sheets

Though you will learn using style sheets with HTML in a separate tutorial, but here you must make a note that if you are using Cascading Style Sheet (CSS) in your HTML code then it is recommended to put that style sheet code inside proper HTML comments so that old browsers can work properly.

### Example

[Live Demo](http://tpcg.io/jULPQO)

<!DOCTYPE html>

<html>

<head>

<title>Commenting Style Sheets</title>

<style>

<!--

.example {

border:1px solid #4a7d49;

}

//-->

</style>

</head>

<body>

<div class = "example">Hello , World!</div>

</body>

</html>

This will produce the following result −

# HTML Images

[❮ Previous](https://www.w3schools.com/html/html_links_bookmarks.asp)[Next ❯](https://www.w3schools.com/html/html_images_imagemap.asp)

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

### Example

<img src="pic\_trulli.jpg" alt="Italian Trulli">

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_trulli)

### Example

<img src="img\_girl.jpg" alt="Girl in a jacket">

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_girl)

### Example

<img src="img\_chania.jpg" alt="Flowers in Chania">

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_chania)

## HTML Images Syntax

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

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

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

<img src="*url*">

## The alt Attribute

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

The value of the alt attribute should describe the image:

### Example

<img src="img\_chania.jpg" alt="Flowers in Chania">

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_alt_chania)

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_wrongname)

**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

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

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_size)

Alternatively, you can use the width and height attributes:

### Example

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

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_attributes)

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 HTML.

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

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_style)

## Images in Another Folder

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

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_folder)

## Images on Another Server

Some web sites store their images on image servers.

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

### Example

<img src="https://www.w3schools.com/images/w3schools\_green.jpg" alt="W3Schools.com">

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_w3schools)

You can read more about file paths in the chapter [HTML File Paths](https://www.w3schools.com/html/html_filepaths.asp).

## Animated Images

HTML allows animated GIFs:

### Example

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

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_hackman)

## Image as a Link

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_link)

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

## Image Floating

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/html/tryit.asp?filename=tryhtml_images_float)

**Tip:** To learn more about CSS Float, read our [CSS Float Tutorial](https://www.w3schools.com/css/css_float.asp).

# HTML Form

An **HTML form** is *a section of a document* which contains controls such as text fields, password fields, checkboxes, radio buttons, submit button, menus etc.

An HTML form facilitates the user to enter data that is to be sent to the server for processing such as name, email address, password, phone number, etc. .

## Why use HTML Form

HTML forms are required if you want to collect some data from of the site visitor.

For example: If a user want to purchase some items on internet, he/she must fill the form such as shipping address and credit/debit card details so that item can be sent to the given address.

## HTML Form Syntax

1. **<form** action="server url" method="get|post"**>**
2. //input controls e.g. textfield, textarea, radiobutton, button
3. **</form>**

## HTML Form Tags

Let's see the list of HTML 5 form tags.

|  |  |
| --- | --- |
| **Tag** | **Description** |
| <form> | It defines an HTML form to enter inputs by the used side. |
| <input> | It defines an input control. |
| <textarea> | It defines a multi-line input control. |
| <label> | It defines a label for an input element. |
| <fieldset> | It groups the related element in a form. |
| <legend> | It defines a caption for a <fieldset> element. |
| <select> | It defines a drop-down list. |
| <optgroup> | It defines a group of related options in a drop-down list. |
| <option> | It defines an option in a drop-down list. |
| <button> | It defines a clickable button. |

## HTML 5 Form Tags

Let's see the list of HTML 5 form tags.

|  |  |
| --- | --- |
| **Tag** | **Description** |
| <datalist> | It specifies a list of pre-defined options for input control. |
| <keygen> | It defines a key-pair generator field for forms. |
| <output> | It defines the result of a calculation. |

## HTML <form> element

The HTML <form> element provide a document section to take input from user. It provides various interactive controls for submitting information to web server such as text field, text area, password field, etc.

#### Note: The <form> element does not itself create a form but it is container to contain all required form elements, such as <input>, <label>, etc.

**Syntax:**

1. **<form>**
2. //Form elements
3. **</form>**

## HTML <input> element

The HTML <input> element is fundamental form element. It is used to create form fields, to take input from user. We can apply different input filed to gather different information form user. Following is the example to show the simple text input.

## Example:

1. **<body>**
2. **<form>**
3. Enter your name  **<br>**
4. **<input** type="text" name="username"**>**
5. **</form>**
6. **</body>**

**Output:**

## HTML TextField Control

The type="text" attribute of input tag creates textfield control also known as single line textfield control. The name attribute is optional, but it is required for the server side component such as JSP, ASP, PHP etc.

1. **<form>**
2. First Name: **<input** type="text" name="firstname"**/>** **<br/>**
3. Last Name:  **<input** type="text" name="lastname"**/>** **<br/>**
4. **</form>**

**Output:**

#### Note: If you will omit 'name' attribute then the text filed input will not be submitted to server.

## HTML <textarea> tag in form

The <textarea> tag in HTML is used to insert multiple-line text in a form. The size of <textarea> can be specify either using "rows" or "cols" attribute or by CSS.

**Example:**

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<title>**Form in HTML**</title>**
5. **</head>**
6. **<body>**
7. **<form>**
8. Enter your address:**<br>**
9. **<textarea** rows="2" cols="20"**></textarea>**
10. **</form>**
11. **</body>**
12. **</html>**

**Output:**

## Label Tag in Form

It is considered better to have label in form. As it makes the code parser/browser/user friendly.

If you click on the label tag, it will focus on the text control. To do so, you need to have for attribute in label tag that must be same as id attribute of input tag.

#### NOTE: It is good to use <label> tag with form, although it is optional but if you will use it, then it will provide a focus when you tap or click on label tag. It is more worthy with touchscreens.

1. **<form>**
2. **<label** for="firstname"**>**First Name: **</label>** **<br/>**
3. **<input** type="text" id="firstname" name="firstname"**/>** **<br/>**
4. **<label** for="lastname"**>**Last Name: **</label>**
5. **<input** type="text" id="lastname" name="lastname"**/>** **<br/>**
6. **</form>**

**Output:**

## HTML Password Field Control

The password is not visible to the user in password field control.

1. **<form>**
2. **<label** for="password"**>**Password: **</label>**
3. **<input** type="password" id="password" name="password"**/>** **<br/>**
4. **</form>**

**Output:**

## HTML 5 Email Field Control

The email field in new in HTML 5. It validates the text for correct email address. You must use @ and . in this field.

1. **<form>**
2. **<label** for="email"**>**Email: **</label>**
3. **<input** type="email" id="email" name="email"**/>** **<br/>**
4. **</form>**

It will display in browser like below:

#### Note: If we will not enter the correct email, it will display error like:

## Radio Button Control

The radio button is used to select one option from multiple options. It is used for selection of gender, quiz questions etc.

If you use one name for all the radio buttons, only one radio button can be selected at a time.

Using radio buttons for multiple options, you can only choose a single option at a time.

1. **<form>**
2. **<label** for="gender"**>**Gender: **</label>**
3. **<input** type="radio" id="gender" name="gender" value="male"**/>**Male
4. **<input** type="radio" id="gender" name="gender" value="female"**/>**Female **<br/>**
5. **</form>**

## Checkbox Control

The checkbox control is used to check multiple options from given checkboxes.

1. **<form>**
2. Hobby:**<br>**
3. **<input** type="checkbox" id="cricket" name="cricket" value="cricket"**/>**
4. **<label** for="cricket"**>**Cricket**</label>** **<br>**
5. **<input** type="checkbox" id="football" name="football" value="football"**/>**
6. **<label** for="football"**>**Football**</label>** **<br>**
7. **<input** type="checkbox" id="hockey" name="hockey" value="hockey"**/>**
8. **<label** for="hockey"**>**Hockey**</label>**
9. **</form>**

#### Note: These are similar to radio button except it can choose multiple options at a time and radio button can select one button at a time, and its display.

**Output:**

## Submit button control

HTML **<input type="submit">** are used to add a submit button on web page. When user clicks on submit button, then form get submit to the server.

Syntax:

1. **<input** type="submit" value="submit"**>**

The type = submit , specifying that it is a submit button

The value attribute can be anything which we write on button on web page.

The name attribute can be omit here.

**Example:**

1. **<form>**
2. **<label** for="name"**>**Enter name**</label><br>**
3. **<input** type="text" id="name" name="name"**><br>**
4. **<label** for="pass"**>**Enter Password**</label><br>**
5. **<input** type="Password" id="pass" name="pass"**><br>**
6. **<input** type="submit" value="submit"**>**
7. **</form>**

**Output:**

## HTML <fieldset> element:

The <fieldset> element in HTML is used to group the related information of a form. This element is used with <legend> element which provide caption for the grouped elements.

**Example:**

1. **<form>**
2. **<fieldset>**
3. **<legend>**User Information:**</legend>**
4. **<label** for="name"**>**Enter name**</label><br>**
5. **<input** type="text" id="name" name="name"**><br>**
6. **<label** for="pass"**>**Enter Password**</label><br>**
7. **<input** type="Password" id="pass" name="pass"**><br>**
8. **<input** type="submit" value="submit"**>**
9. **</fieldset>**
10. lt;/form**>**

**Output:**

## HTML Form Example

Following is the example for a simple form of registration.

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<title>**Form in HTML**</title>**
5. **</head>**
6. **<body>**
7. **<h2>**Registration form**</h2>**
8. **<form>**
9. **<fieldset>**
10. **<legend>**User personal information**</legend>**
11. **<label>**Enter your full name**</label><br>**
12. **<input** type="text" name="name"**><br>**
13. **<label>**Enter your email**</label><br>**
14. **<input** type="email" name="email"**><br>**
15. **<label>**Enter your password**</label><br>**
16. **<input** type="password" name="pass"**><br>**
17. **<label>**confirm your password**</label><br>**
18. **<input** type="password" name="pass"**><br>**
19. **<br><label>**Enter your gender**</label><br>**
20. **<input** type="radio" id="gender" name="gender" value="male"**/>**Male  **<br>**
21. **<input** type="radio" id="gender" name="gender" value="female"**/>**Female **<br/>**
22. **<input** type="radio" id="gender" name="gender" value="others"**/>**others **<br/>**
23. **<br>**Enter your Address:**<br>**
24. **<textarea></textarea><br>**
25. **<input** type="submit" value="sign-up"**>**
26. **</fieldset>**
27. **</form>**
28. **</body>**
29. **</html>**

**[Test it Now](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlform2" \t "_blank)**

**Output:**

## HTML Form Example

Let's see a simple example of creating HTML form.

1. **<form** action="#"**>**
2. **<table>**
3. **<tr>**
4. **<td** class="tdLabel"**><label** for="register\_name" class="label"**>**Enter name:**</label></td>**
5. **<td><input** type="text" name="name" value="" id="register\_name" style="width:160px"**/></td>**
6. **</tr>**
7. **<tr>**
8. **<td** class="tdLabel"**><label** for="register\_password" class="label"**>**Enter password:**</label></td>**
9. **<td><input** type="password" name="password" id="register\_password" style="width:160px"**/></td>**
10. **</tr>**
11. **<tr>**
12. **<td** class="tdLabel"**><label** for="register\_email" class="label"**>**Enter Email:**</label></td>**
13. **<td**
14. **><input** type="email" name="email" value="" id="register\_email" style="width:160px"**/></td>**
15. **</tr>**
16. **<tr>**
17. **<td** class="tdLabel"**><label** for="register\_gender" class="label"**>**Enter Gender:**</label></td>**
18. **<td>**
19. **<input** type="radio" name="gender" id="register\_gendermale" value="male"**/>**
20. **<label** for="register\_gendermale"**>**male**</label>**
21. **<input** type="radio" name="gender" id="register\_genderfemale" value="female"**/>**
22. **<label** for="register\_genderfemale"**>**female**</label>**
23. **</td>**
24. **</tr>**
25. **<tr>**
26. **<td** class="tdLabel"**><label** for="register\_country" class="label"**>**Select Country:**</label></td>**
27. **<td><select** name="country" id="register\_country" style="width:160px"**>**
28. **<option** value="india"**>**india**</option>**
29. **<option** value="pakistan"**>**pakistan**</option>**
30. **<option** value="africa"**>**africa**</option>**
31. **<option** value="china"**>**china**</option>**
32. **<option** value="other"**>**other**</option>**
33. **</select>**
34. **</td>**
35. **</tr>**
36. **<tr>**
37. **<td** colspan="2"**><div** align="right"**><input** type="submit" id="register\_0" value="register"**/>**
38. **</div></td>**
39. **</tr>**
40. **</table>**
41. **</form>**

# HTML Form Input Types

In HTML <input type=" "> is an important element of HTML form. The "type" attribute of input element can be various types, which defines information field. Such as <input type="text" name="name"> gives a text box.

## Following is a list of all types of <input> element of HTML.

|  |  |
| --- | --- |
| **type=" "** | **Description** |
| text | Defines a one-line text input field |
| password | Defines a one-line password input field |
| submit | Defines a submit button to submit the form to server |
| reset | Defines a reset button to reset all values in the form. |
| radio | Defines a radio button which allows select one option. |
| checkbox | Defines checkboxes which allow select multiple options form. |
| button | Defines a simple push button, which can be programmed to perform a task on an event. |
| file | Defines to select the file from device storage. |
| image | Defines a graphical submit button. |

**HTML5 added new types on <input> element. Following is the list of types of elements of HTML5**

|  |  |
| --- | --- |
| **type=" "** | **Description** |
| color | Defines an input field with a specific color. |
| date | Defines an input field for selection of date. |
| datetime-local | Defines an input field for entering a date without time zone. |
| email | Defines an input field for entering an email address. |
| month | Defines a control with month and year, without time zone. |
| number | Defines an input field to enter a number. |
| url | Defines a field for entering URL |
| week | Defines a field to enter the date with week-year, without time zone. |
| search | Defines a single line text field for entering a search string. |
| tel | Defines an input field for entering the telephone number. |

**Following is the description about types of <input> element with examples.**

### 1. <input type="text">:

<input> element of type "text" are used to define a single-line input text field.

### Example:

1. **<form>**
2. **<label>**Enter first name**</label><br>**
3. **<input** type="text" name="firstname"**><br>**
4. **<label>**Enter last name**</label><br>**
5. **<input** type="text" name="lastname"**><br>**
6. **<p><strong>**Note:**</strong>**The default maximum cahracter lenght is 20.**</p>**
7. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes)

**Output:**

### Input "text" type:

The **"text"**field defines a sinlge line input text field.

Top of Form

Enter first name  
  
Enter last name  


**Note:**The default maximum cahracter lenght is 20.

Bottom of Form

### 2. <input type="password">:

The <input> element of type "password" allow a user to enter the password securely in a webpage. The entered text in password filed converted into "\*" or ".", so that it cannot be read by another user.

### Example:

1. **<form>**
2. **<label>**Enter User name**</label><br>**
3. **<input** type="text" name="firstname"**><br>**
4. **<label>**Enter Password**</label><br>**
5. **<input** type="Password" name="password"**><br>**
6. **<br><input** type="submit" value="submit"**>**
7. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes2)

**Output:**

### Input "password" type:

The **"password"**field defines a sinlge line input password field to enter the password securely.

Top of Form

Enter User name  
  
Enter Password  
  
  


Bottom of Form

### 3. <input type="submit">:

The <input> element of type "submit" defines a submit button to submit the form to the server when the "click" event occurs.

### Example:

1. **<form** action="https://www.javatpoint.com/html-tutorial"**>**
2. **<label>**Enter User name**</label><br>**
3. **<input** type="text" name="firstname"**><br>**
4. **<label>**Enter Password**</label><br>**
5. **<input** type="Password" name="password"**><br>**
6. **<br><input** type="submit" value="submit"**>**
7. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes3)

**Output:**

### Input "submit" type:

Top of Form

Enter User name  
  
Enter Password  
  
  


Bottom of Form

After clicking on submit button, this will submit the form to server and will redirect the page to **action**value.We will learn about "action" attribute in later chapters

### 4. <input type="reset">:

The <input> type "reset" is also defined as a button but when the user performs a click event, it by default reset the all inputted values.

### Example:

1. **<form>**
2. **<label>**User id: **</label>**
3. **<input** type="text" name="user-id" value="user"**>**
4. **<label>**Password: **</label>**
5. **<input** type="password" name="pass" value="pass"**><br><br>**
6. **<input** type="submit" value="login"**>**
7. **<input** type="reset" value="Reset"**>**
8. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes4)

**Output:**

### Input "reset" type:

Top of Form

User id:  Password:   
  
 

Bottom of Form

Try to change the input values of user id and password, then when you click on reset, it will reset input fields with default values.

### 5. <input type="radio">:

The <input> type "radio" defines the radio buttons, which allow choosing an option between a set of related options. At a time only one radio button option can be selected at a time.

### Example:

1. **<form>**
2. **<p>**Kindly Select your favorite color**</p>**
3. **<input** type="radio" name="color" value="red"**>** Red **<br>**
4. **<input** type="radio" name="color" value="blue"**>** blue **<br>**
5. **<input** type="radio" name="color" value="green"**>**green **<br>**
6. **<input** type="radio" name="color" value="pink"**>**pink **<br>**
7. **<input** type="submit" value="submit"**>**
8. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes5)

**Output:**

### Input "radio" type

Top of Form

Kindly Select your favorite color

 Red  
 blue  
green  
pink  


Bottom of Form

### 6. <input type="checkbox">:

The <input> type "checkbox" are displayed as square boxes which can be checked or unchecked to select the choices from the given options.

#### Note: The "radio" buttons are similar to checkboxes, but there is an important difference between both types: radio buttons allow the user to select only one option at a time, whereas checkbox allows a user to select zero to multiple options at a time.

### Example:

1. **<form>**
2. **<label>**Enter your Name:**</label>**
3. **<input** type="text" name="name"**>**
4. **<p>**Kindly Select your favourite sports**</p>**
5. **<input** type="checkbox" name="sport1" value="cricket"**>**Cricket**<br>**
6. **<input** type="checkbox" name="sport2" value="tennis"**>**Tennis**<br>**
7. **<input** type="checkbox" name="sport3" value="football"**>**Football**<br>**
8. **<input** type="checkbox" name="sport4" value="baseball"**>**Baseball**<br>**
9. **<input** type="checkbox" name="sport5" value="badminton"**>**Badminton**<br><br>**
10. **<input** type="submit" value="submit"**>**
11. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes6)

**Output:**

## Input "checkbox" type

### Registration Form

Top of Form

Enter your Name: 

Kindly Select your favorite sports

Cricket  
Tennis  
Football  
Baseball  
Badminton  
  


Bottom of Form

### 7. <input type="button">:

The <input> type "button" defines a simple push button, which can be programmed to control a functionally on any event such as, click event.

#### Note: It mainly works with JavaScript.

### Example:

1. **<form>**
2. **<input** type="button" value="Clcik me " onclick="alert('you are learning HTML')"**>**
3. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes7)

**Output:**

## Input "button" type.

Click the button to see the result:

Top of Form

Bottom of Form

#### Note: In the above example we have used the "alert" of JS, which you will learn in our JS tutorial. It is used to show a pop window.

### 8. <input type="file">:

The <input> element with type "file" is used to select one or more files from user device storage. Once you select the file, and after submission, this file can be uploaded to the server with the help of JS code and file API.

### Example:

1. **<form>**
2. **<label>**Select file to upload:**</label>**
3. **<input** type="file" name="newfile"**>**
4. **<input** type="submit" value="submit"**>**
5. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes8)

**Output:**

## Input "file" type.

We can choose any type of file until we do not specify it! The selected file will appear at next to "choose file" option

Top of Form

Select file to upload:  

Bottom of Form

### 9. <input type="image">:

The <input> type "image" is used to represent a submit button in the form of image.

### Example:

1. <!DOCTYPE html**>**
2. **<html>**
3. **<body>**
4. **<h2>**Input "image" type.**</h2>**
5. **<p>**We can create an image as submit button**</p>**
6. **<form>**
7. **<label>**User id:**</label><br>**
8. **<input** type="text" name="name"**><br><br>**
9. **<input** type="image" alt="Submit" src="login.png"  width="100px"**>**
10. **</form>**
12. **</body>**
13. **</html>**

## HTML5 newly added <input> types element

### 1. <input type="color">:

The <input> type "color" is used to define an input field which contains a colour. It allows a user to specify the colour by the visual colour interface on a browser.

#### Note: The "color" type only supports color value in hexadecimal format, and the default value is #000000 (black).

### Example:

1. **<form>**
2. Pick your Favorite color: **<br><br>**
3. **<input** type="color" name="upclick" value="#a52a2a"**>** Upclick**<br><br>**
4. **<input** type="color" name="downclick" value="#f5f5dc"**>** Downclick
5. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes1_1)

**Output:**

### Input "color" types:

Top of Form

Pick your Favorite color:  
  
 Up-click  
  
 Down-click

Bottom of Form

**Note:**The default value of "color" type is #000000 (black). It only supports color value in hexadecimal format.

### 2. <input type="date">:

The <input> element of type "date" generates an input field, which allows a user to input the date in a given format. A user can enter the date by text field or by date picker interface.

### Example:

1. **<form>**
2. Select Start and End Date: **<br><br>**
3. **<input** type="date" name="Startdate"**>** Start date:**<br><br>**
4. **<input** type="date" name="Enddate"**>** End date:**<br><br>**
5. **<input** type="submit"**>**
6. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes2_2)

**Output:**

### Input "date" type

Top of Form

Select Start and End Date:  
  
 Start date:  
  
 End date:  
  


Bottom of Form

### 3. <input type="datetime-local">:

The <input> element of type "datetime-local" creates input filed which allow a user to select the date as well as local time in the hour and minute without time zone information.

### Example:

1. **<form>**
2. **<label>**
3. Select the meeting schedule: **<br><br>**
4. Select date & time: **<input** type="datetime-local" name="meetingdate"**>** **<br><br>**
5. **</label>**
6. **<input** type="submit"**>**
7. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes3_3)

**Output:**

### Input "datetime-local" type

Top of Form

Select the meeting schedule:  
  
Select date & time:   
  


Bottom of Form

### 4. <input type="email">:

The <input> type "email" creates an input filed which allow a user to enter the e-mail address with pattern validation. The multiple attributes allow a user to enter more than one email address.

### Example:

1. **<form>**
2. **<label><b>**Enter your Email-address**</b></label>**
3. **<input** type="email" name="email" required**>**
4. **<input** type="submit"**>**
5. **<p><strong>**Note:**</strong>**User can also enter multiple email addresses separating by comma or whitespace as following: **</p>**
6. **<label><b>**Enter multiple Email-addresses**</b></label>**
7. **<input** type="email" name="email"  multiple**>**
8. **<input** type="submit"**>**
9. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes4_4)

**Output:**

### Input "email" type

Top of Form

**Enter your Email-address**  

**Note:**User can also enter multiple email addresses separating by comma or whitespace as following:

**Enter multiple Email-addresses**  

Bottom of Form

### 5. <input type="month">:

The <input> type "month" creates an input field which allows a user to easily enter month and year in the format of "MM, YYYY" where MM defines month value, and YYYY defines the year value. New

### Example:

1. **<form>**
2. **<label>**Enter your Birth Month-year: **</label>**
3. **<input** type="month" name="newMonth"**>**
4. **<input** type="submit"**>**
5. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes5_5)

**Output:**

### Input "month" type:

Top of Form

Enter your Birth Month-year:  

Bottom of Form

### 6. <input type="number">:

The <input> element type number creates input filed which allows a user to enter the numeric value. You can also restrict to enter a minimum and maximum value using min and max attribute.

### Example:

1. **<form>**
2. **<label>**Enter your age: **</label>**
3. **<input** type="number" name="num" min="50" max="80"**>**
4. **<input** type="submit"**>**
5. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes6_6)

**Output:**

### Input "number" type

Top of Form

Enter your age:  

Bottom of Form

**Note:**It will allow to enter number in range of 50-80. If you want to enter number other than range, it will show an error.

### 7. <input type="url">:

The <input> element of type "url" creates an input filed which enables user to enter the URL.

### Example:

1. **<form>**
2. **<label>**Enter your website URL: **</label>**
3. **<input** type="url" name="website" placeholder="http://example.com"**><br>**
4. **<input** type="submit" value="send data"**>**
5. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes7_7)

**Output:**

### Input "url" type

Top of Form

Enter your website URL:   


Bottom of Form

### 8. <input type="week">:

The <input> type week creates an input field which allows a user to select a week and year form the drop-down calendar without time zone.

### Example:

1. **<form>**
2. **<label><b>**Select your best week of year:**</b></label><br><br>**
3. **<input** type="week" name="bestweek"**>**
4. **<input** type="submit" value="Send data"**>**
5. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes8_8)

**Output:**

## Input "week" type

Top of Form

**Select your best week of year:**  
  
 

Bottom of Form

### 9. <input type="search">:

The <input> type "search" creates an input filed which allows a user to enter a search string. These are functionally symmetrical to the text input type, but may be styled differently.

### Example:

1. **<form>**
2. **<label>**Search here:**</label>**
3. **<input** type="search" name="q"**>**
4. **<input** type="submit" value="search"**>**
5. **</form>**

[**Test it Now**](http://www.javatpoint.com/oprweb/test.jsp?filename=htmlFormInputTypes9_9)

**Output:**

## Input "search" type

Top of Form

Search here:  

Bottom of Form

### 10. <input type="tel">:

The <input> element of type ?tel? creates an input filed to enter the telephone number. The "tel" type does not have default validation such as email, because telephone number pattern can vary worldwide.

### Example:

1. **<form>**
2. **<label><b>**Enter your Telephone Number(in format of xxx-xxx-xxxx):**</b></label>**
3. **<input** type="tel" name="telephone" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}" required**>**
4. **<input** type="submit"**><br><br>**
5. **</form>**

**CSS** is used to control the style of a web document in a simple and easy way.

**CSS** is the acronym for **"Cascading Style Sheet"**. This tutorial covers both the versions CSS1,CSS2 and CSS3, and gives a complete understanding of CSS, starting from its basics to advanced concepts.

## Why to Learn CSS?

**Cascading Style Sheets**, fondly referred to as **CSS**, is a simple design language intended to simplify the process of making web pages presentable.

**CSS** is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning CSS:

* **Create Stunning Web site** - CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs,variations in display for different devices and screen sizes as well as a variety of other effects.
* **Become a web designer** - If you want to start a carrer as a professional web designer, HTML and CSS designing is a must skill.
* **Control web** - CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.
* **Learn other languages** - Once you understands the basic of HTML and CSS then other related technologies like javascript, php, or angular are become easier to understand.

## Hello World using CSS.

Just to give you a little excitement about CSS, I'm going to give you a small conventional CSS Hello World program, You can try it using Demo link.

[Live Demo](http://tpcg.io/qfPFF4)

<!DOCTYPE html>

<html>

<head>

<title>This is document title</title>

<style>

h1 {

color: #36CFFF;

}

</style>

</head>

<body>

<h1>Hello World!</h1>

</body>

</html>

## Applications of CSS

As mentioned before, CSS is one of the most widely used style language over the web. I'm going to list few of them here:

* **CSS saves time** - You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
* **Pages load faster** - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
* **Easy maintenance** - To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
* **Superior styles to HTML** - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
* **Multiple Device Compatibility** - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
* **Global web standards**- Now HTML attributes are being deprecated and it is being recommended to use CSS. So its a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

**C**ascading **S**tyle **S**heets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs,variations in display for different devices and screen sizes as well as a variety of other effects.

CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

Advantages of CSS

* **CSS saves time** − You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
* **Pages load faster** − If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
* **Easy maintenance** − To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
* **Superior styles to HTML** − CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
* **Multiple Device Compatibility** − Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
* **Global web standards** − Now HTML attributes are being deprecated and it is being recommended to use CSS. So its a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

Who Creates and Maintains CSS?

CSS is created and maintained through a group of people within the W3C called the CSS Working Group. The CSS Working Group creates documents called specifications. When a specification has been discussed and officially ratified by the W3C members, it becomes a recommendation.

These ratified specifications are called recommendations because the W3C has no control over the actual implementation of the language. Independent companies and organizations create that software.

**NOTE** − The World Wide Web Consortium, or W3C is a group that makes recommendations about how the Internet works and how it should evolve.

CSS Versions

Cascading Style Sheets level 1 (CSS1) came out of W3C as a recommendation in December 1996. This version describes the CSS language as well as a simple visual formatting model for all the HTML tags.

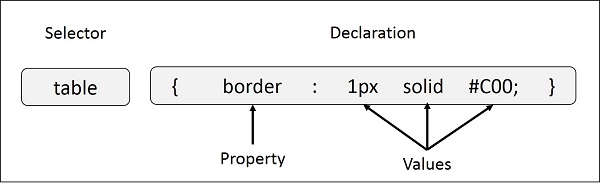
CSS2 became a W3C recommendation in May 1998 and builds on CSS1. This version adds support for media-specific style sheets e.g. printers and aural devices, downloadable fonts, element positioning and tables.

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts −

* **Selector** − A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.
* **Property** − A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be *color*, *border* etc.
* **Value** − Values are assigned to properties. For example, *color* property can have value either *red* or *#F1F1F1* etc.

You can put CSS Style Rule Syntax as follows −

selector { property: value }



**Example** − You can define a table border as follows −

table{ border :1px solid #C00; }

Here table is a selector and border is a property and given value *1px solid #C00* is the value of that property.

You can define selectors in various simple ways based on your comfort. Let me put these selectors one by one.

The Type Selectors

This is the same selector we have seen above. Again, one more example to give a color to all level 1 headings −

h1 {

color: #36CFFF;

}

The Universal Selectors

Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type −

\* {

color: #000000;

}

This rule renders the content of every element in our document in black.

The Descendant Selectors

Suppose you want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, style rule will apply to <em> element only when it lies inside <ul> tag.

ul em {

color: #000000;

}

The Class Selectors

You can define style rules based on the class attribute of the elements. All the elements having that class will be formatted according to the defined rule.

.black {

color: #000000;

}

This rule renders the content in black for every element with class attribute set to *black* in our document. You can make it a bit more particular. For example −

h1.black {

color: #000000;

}

This rule renders the content in black for only <h1> elements with class attribute set to *black*.

You can apply more than one class selectors to given element. Consider the following example −

<p class = "center bold">

This para will be styled by the classes *center* and *bold*.

</p>

The ID Selectors

You can define style rules based on the *id* attribute of the elements. All the elements having that *id* will be formatted according to the defined rule.

#black {

color: #000000;

}

This rule renders the content in black for every element with *id* attribute set to *black* in our document. You can make it a bit more particular. For example −

h1#black {

color: #000000;

}

This rule renders the content in black for only <h1> elements with *id* attribute set to *black*.

The true power of *id* selectors is when they are used as the foundation for descendant selectors, For example −

#black h2 {

color: #000000;

}

In this example all level 2 headings will be displayed in black color when those headings will lie with in tags having *id* attribute set to *black*.

The Child Selectors

You have seen the descendant selectors. There is one more type of selector, which is very similar to descendants but have different functionality. Consider the following example −

body > p {

color: #000000;

}

This rule will render all the paragraphs in black if they are direct child of <body> element. Other paragraphs put inside other elements like <div> or <td> would not have any effect of this rule.

The Attribute Selectors

You can also apply styles to HTML elements with particular attributes. The style rule below will match all the input elements having a type attribute with a value of *text* −

input[type = "text"] {

color: #000000;

}

The advantage to this method is that the <input type = "submit" /> element is unaffected, and the color applied only to the desired text fields.

There are following rules applied to attribute selector.

* **p[lang]** − Selects all paragraph elements with a *lang* attribute.
* **p[lang="fr"]** − Selects all paragraph elements whose *lang* attribute has a value of exactly "fr".
* **p[lang~="fr"]** − Selects all paragraph elements whose *lang* attribute contains the word "fr".
* **p[lang|="en"]** − Selects all paragraph elements whose *lang* attribute contains values that are exactly "en", or begin with "en-".

Multiple Style Rules

You may need to define multiple style rules for a single element. You can define these rules to combine multiple properties and corresponding values into a single block as defined in the following example −

h1 {

color: #36C;

font-weight: normal;

letter-spacing: .4em;

margin-bottom: 1em;

text-transform: lowercase;

}

Here all the property and value pairs are separated by a **semicolon (;)**. You can keep them in a single line or multiple lines. For better readability, we keep them in separate lines.

For a while, don't bother about the properties mentioned in the above block. These properties will be explained in the coming chapters and you can find complete detail about properties in CSS References

Grouping Selectors

You can apply a style to many selectors if you like. Just separate the selectors with a comma, as given in the following example −

h1, h2, h3 {

color: #36C;

font-weight: normal;

letter-spacing: .4em;

margin-bottom: 1em;

text-transform: lowercase;

}

This define style rule will be applicable to h1, h2 and h3 element as well. The order of the list is irrelevant. All the elements in the selector will have the corresponding declarations applied to them.

You can combine the various *id* selectors together as shown below −

#content, #footer, #supplement {

position: absolute;

left: 510px;

width: 200px;

}

There are four ways to associate styles with your HTML document. Most commonly used methods are inline CSS and External CSS.

## Embedded CSS - The <style> Element

You can put your CSS rules into an HTML document using the <style> element. This tag is placed inside the <head>...</head> tags. Rules defined using this syntax will be applied to all the elements available in the document. Here is the generic syntax −

[Live Demo](http://tpcg.io/ow2sPD)

[Live Demo](http://tpcg.io/ow2sPD)

<!DOCTYPE html>

<html>

<head>

<style type = "text/css" media = "all">

body {

background-color: linen;

}

h1 {

color: maroon;

margin-left: 40px;

}

</style>

</head>

<body>

<h1>This is a heading</h1>

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

</body>

</html>

It will produce the following result −

### Attributes

Attributes associated with <style> elements are −

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Description** |
| type | text/css | Specifies the style sheet language as a content-type (MIME type). This is required attribute. |
| media | screen  tty  tv  projection  handheld  print  braille  aural  all | Specifies the device the document will be displayed on. Default value is *all*. This is an optional attribute. |

## Inline CSS - The *style* Attribute

You can use *style* attribute of any HTML element to define style rules. These rules will be applied to that element only. Here is the generic syntax −

<element style = "...style rules....">

### Attributes

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Description** |
| style | style rules | The value of *style* attribute is a combination of style declarations separated by semicolon (;). |

### Example

Following is the example of inline CSS based on the above syntax −

[Live Demo](http://tpcg.io/tDUIUL)

<html>

<head>

</head>

<body>

<h1 style = "color:#36C;">

This is inline CSS

</h1>

</body>

</html>

It will produce the following result −

## External CSS - The <link> Element

The <link> element can be used to include an external stylesheet file in your HTML document.

An external style sheet is a separate text file with **.css** extension. You define all the Style rules within this text file and then you can include this file in any HTML document using <link> element.

Here is the generic syntax of including external CSS file −

<head>

<link type = "text/css" href = "..." media = "..." />

</head>

### Attributes

Attributes associated with <style> elements are −

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Description** |
| type | text css | Specifies the style sheet language as a content-type (MIME type). This attribute is required. |
| href | URL | Specifies the style sheet file having Style rules. This attribute is a required. |
| media | screen  tty  tv  projection  handheld  print  braille  aural  all | Specifies the device the document will be displayed on. Default value is *all*. This is optional attribute. |

### Example

Consider a simple style sheet file with a name *mystyle.css* having the following rules −

h1, h2, h3 {

color: #36C;

font-weight: normal;

letter-spacing: .4em;

margin-bottom: 1em;

text-transform: lowercase;

}

Now you can include this file *mystyle.css* in any HTML document as follows −

<head>

<link type = "text/css" href = "mystyle.css" media = " all" />

</head>

## Imported CSS - @import Rule

@import is used to import an external stylesheet in a manner similar to the <link> element. Here is the generic syntax of @import rule.

<head>

<@import "URL";

</head>

Here URL is the URL of the style sheet file having style rules. You can use another syntax as well −

<head>

<@import url("URL");

</head>

### Example

Following is the example showing you how to import a style sheet file into HTML document −

<head>

@import "mystyle.css";

</head>

## CSS Rules Overriding

We have discussed four ways to include style sheet rules in a an HTML document. Here is the rule to override any Style Sheet Rule.

* Any inline style sheet takes highest priority. So, it will override any rule defined in <style>...</style> tags or rules defined in any external style sheet file.
* Any rule defined in <style>...</style> tags will override rules defined in any external style sheet file.
* Any rule defined in external style sheet file takes lowest priority, and rules defined in this file will be applied only when above two rules are not applicable.

## Handling old Browsers

There are still many old browsers who do not support CSS. So, we should take care while writing our Embedded CSS in an HTML document. The following snippet shows how you can use comment tags to hide CSS from older browsers −

<style type = "text/css">

<!--

body, td {

color: blue;

}

-->

</style>

## CSS Comments

Many times, you may need to put additional comments in your style sheet blocks. So, it is very easy to comment any part in style sheet. You can simple put your comments inside /\*.....this is a comment in style sheet.....\*/.

You can use /\* ....\*/ to comment multi-line blocks in similar way you do in C and C++ programming languages.

### Example

[Live Demo](http://tpcg.io/P2Uj8x)

<!DOCTYPE html>

<html>

<head>

<style>

p {

color: red;

/\* This is a single-line comment \*/

text-align: center;

}

/\* This is a multi-line comment \*/

</style>

</head>

<body>

<p>Hello World!</p>

</body>

</html>

It will produce the following result −

Before we start the actual exercise, we would like to give a brief idea about the CSS Measurement Units. CSS supports a number of measurements including absolute units such as inches, centimeters, points, and so on, as well as relative measures such as percentages and em units. You need these values while specifying various measurements in your Style rules e.g. **border = "1px solid red"**.

We have listed out all the CSS Measurement Units along with proper Examples −

|  |  |  |
| --- | --- | --- |
| **Unit** | **Description** | **Example** |
| % | Defines a measurement as a percentage relative to another value, typically an enclosing element. | p {font-size: 16pt; line-height: 125%;} |
| cm | Defines a measurement in centimeters. | div {margin-bottom: 2cm;} |
| em | A relative measurement for the height of a font in em spaces. Because an em unit is equivalent to the size of a given font, if you assign a font to 12pt, each "em" unit would be 12pt; thus, 2em would be 24pt. | p {letter-spacing: 7em;} |
| ex | This value defines a measurement relative to a font's x-height. The x-height is determined by the height of the font's lowercase letter x. | p {font-size: 24pt; line-height: 3ex;} |
| in | Defines a measurement in inches. | p {word-spacing: .15in;} |
| mm | Defines a measurement in millimeters. | p {word-spacing: 15mm;} |
| pc | Defines a measurement in picas. A pica is equivalent to 12 points; thus, there are 6 picas per inch. | p {font-size: 20pc;} |
| pt | Defines a measurement in points. A point is defined as 1/72nd of an inch. | body {font-size: 18pt;} |
| px | Defines a measurement in screen pixels. | p {padding: 25px;} |

You can set the following background properties of an element −

* The **background-color** property is used to set the background color of an element.
* The **background-image** property is used to set the background image of an element.
* The **background-repeat** property is used to control the repetition of an image in the background.
* The **background-position** property is used to control the position of an image in the background.
* The **background-attachment** property is used to control the scrolling of an image in the background.
* The **background** property is used as a shorthand to specify a number of other background properties.

## Set the Background Color

Following is the example which demonstrates how to set the background color for an element.

[Live Demo](http://tpcg.io/DjPt4l)

<html>

<head>

</head>

<body>

<p **style = "background-color:yellow;"**>

This text has a yellow background color.

</p>

</body>

</html>

This will produce following result −

## Set the Background Image

We can set the background image by calling local stored images as shown below −

[Live Demo](http://tpcg.io/zTzFoS)

<html>

<head>

<style>

body {

background-image: url("/css/images/css.jpg");

background-color: #cccccc;

}

</style>

</head>

<body>

<h1>Hello World!</h1>

</body>

<html>

It will produce the following result −

## Repeat the Background Image

The following example demonstrates how to repeat the background image if an image is small. You can use *no-repeat* value for *background-repeat* property if you don't want to repeat an image, in this case image will display only once.

By default *background-repeat* property will have *repeat* value.

[Live Demo](http://tpcg.io/V9zZtU)

<html>

<head>

<style>

body {

background-image: url("/css/images/css.jpg");

background-repeat: repeat;

}

</style>

</head>

<body>

<p>Tutorials point</p>

</body>

</html>

It will produce the following result −

The following example which demonstrates how to repeat the background image vertically.

[Live Demo](http://tpcg.io/y6EVME)

<html>

<head>

<style>

body {

background-image: url("/css/images/css.jpg");

background-repeat: repeat-y;

}

</style>

</head>

<body>

<p>Tutorials point</p>

</body>

</html>

It will produce the following result −

The following example demonstrates how to repeat the background image horizontally.

[Live Demo](http://tpcg.io/wlHE9n)

<html>

<head>

<style>

body {

background-image: url("/css/images/css.jpg");

background-repeat: repeat-x;

}

</style>

</head>

<body>

<p>Tutorials point</p>

</body>

</html>

It will produce the following result −

## Set the Background Image Position

The following example demonstrates how to set the background image position 100 pixels away from the left side.

[Live Demo](http://tpcg.io/RNI2lf)

<html>

<head>

<style>

body {

background-image: url("/css/images/css.jpg");

background-position:100px;

}

</style>

</head>

<body>

<p>Tutorials point</p>

</body>

</html>

It will produce the following result −

The following example demonstrates how to set the background image position 100 pixels away from the left side and 200 pixels down from the top.

[Live Demo](http://tpcg.io/oqwQvk)

<html>

<head>

<style>

body {

background-image: url("/css/images/css.jpg");

background-position:100px 200px;

}

</style>

</head>

<body>

<p>Tutorials point</p>

</body>

</html>

It will produce the following result −

## Set the Background Attachment

Background attachment determines whether a background image is fixed or scrolls with the rest of the page.

The following example demonstrates how to set the fixed background image.

[Live Demo](http://tpcg.io/cBPDsy)

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-image: url('/css/images/css.jpg');

background-repeat: no-repeat;

background-attachment: fixed;

}

</style>

</head>

<body>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

</body>

</html>

It will produce the following result −

The following example demonstrates how to set the scrolling background image.

[Live Demo](http://tpcg.io/qWupRD)

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-image: url('/css/images/css.jpg');

background-repeat: no-repeat;

background-attachment: fixed;

background-attachment:scroll;

}

</style>

</head>

<body>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

<p>The background-image is fixed. Try to scroll down the page.</p>

</body>

</html>

It will produce the following result −

## Shorthand Property

You can use the *background* property to set all the background properties at once. For example −

<p **style = "background:url(/images/pattern1.gif) repeat fixed;"**>

This parapgraph has fixed repeated background image.

</p>

You can set following font properties of an element −

* The **font-family** property is used to change the face of a font.
* The **font-style** property is used to make a font italic or oblique.
* The **font-variant** property is used to create a small-caps effect.
* The **font-weight** property is used to increase or decrease how bold or light a font appears.
* The **font-size** property is used to increase or decrease the size of a font.
* The **font** property is used as shorthand to specify a number of other font properties.

## Set the Font Family

Following is the example, which demonstrates how to set the font family of an element. Possible value could be any font family name.

[Live Demo](http://tpcg.io/bodklA)

<html>

<head>

</head>

<body>

<p **style = "font-family:georgia,garamond,serif;"**>

This text is rendered in either georgia, garamond, or the

default serif font depending on which font you have at your system.

</p>

</body>

</html>

This will produce following result −

## Set the Font Style

Following is the example, which demonstrates how to set the font style of an element. Possible values are *normal, italic and oblique*.

[Live Demo](http://tpcg.io/5ag0Uy)

<html>

<head>

</head>

<body>

<p **style = "font-style:italic;"**>

This text will be rendered in italic style

</p>

</body>

</html>

This will produce following result −

## Set the Font Variant

The following example demonstrates how to set the font variant of an element. Possible values are *normal and small-caps*.

[Live Demo](http://tpcg.io/gjK7kb)

<html>

<head>

</head>

<body>

<p **style = "font-variant:small-caps;"**>

This text will be rendered as small caps

</p>

</body>

</html>

This will produce following result −

## Set the Font Weight

The following example demonstrates how to set the font weight of an element. The font-weight property provides the functionality to specify how bold a font is. Possible values could be *normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900*.

[Live Demo](http://tpcg.io/HKBdoD)

<html>

<head>

</head>

<body>

<p **style = "font-weight:bold;"**>

This font is bold.

</p>

<p **style = "font-weight:bolder;"**>

This font is bolder.

</p>

<p **style = "font-weight:500;"**>

This font is 500 weight.

</p>

</body>

</html>

This will produce following result −

## Set the Font Size

The following example demonstrates how to set the font size of an element. The font-size property is used to control the size of fonts. Possible values could be *xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger, size in pixels or in %*.

[Live Demo](http://tpcg.io/EBK78e)

<html>

<head>

</head>

<body>

<p **style = "font-size:20px;"**>

This font size is 20 pixels

</p>

<p **style = "font-size:small;"**>

This font size is small

</p>

<p **style = "font-size:large;"**>

This font size is large

</p>

</body>

</html>

This will produce following result −

## Set the Font Size Adjust

The following example demonstrates how to set the font size adjust of an element. This property enables you to adjust the x-height to make fonts more legible. Possible value could be any number.

[Live Demo](http://tpcg.io/YQiR1i)

<html>

<head>

</head>

<body>

<p **style = "font-size-adjust:0.61;"**>

This text is using a font-size-adjust value.

</p>

</body>

</html>

This will produce following result −

## Set the Font Stretch

The following example demonstrates how to set the font stretch of an element. This property relies on the user's computer to have an expanded or condensed version of the font being used.

Possible values could be *normal, wider, narrower, ultra-condensed, extra-condensed, condensed, semi-condensed, semi-expanded, expanded, extra-expanded, ultra-expanded*.

[Live Demo](http://tpcg.io/hknNR1)

<html>

<head>

</head>

<body>

<p **style = "font-stretch:ultra-expanded;"**>

If this doesn't appear to work, it is likely that your computer

doesn't have a <br>condensed or expanded version of the font being used.

</p>

</body>

</html>

This will produce following result −

## Shorthand Property

You can use the *font* property to set all the font properties at once. For example −

[Live Demo](http://tpcg.io/Rsu4Ct)

<html>

<head>

</head>

<body>

<p **style = "font:italic small-caps bold 15px georgia;"**>

Applying all the properties on the text at once.

</p>

</body>

</html>

This will produce following result −

You can set following text properties of an element −

* The **color** property is used to set the color of a text.
* The **direction** property is used to set the text direction.
* The **letter-spacing** property is used to add or subtract space between the letters that make up a word.
* The **word-spacing** property is used to add or subtract space between the words of a sentence.
* The **text-indent** property is used to indent the text of a paragraph.
* The **text-align** property is used to align the text of a document.
* The **text-decoration** property is used to underline, overline, and strikethrough text.
* The **text-transform** property is used to capitalize text or convert text to uppercase or lowercase letters.
* The **white-space** property is used to control the flow and formatting of text.
* The **text-shadow** property is used to set the text shadow around a text.

## Set the Text Color

The following example demonstrates how to set the text color. Possible value could be any color name in any valid format.

[Live Demo](http://tpcg.io/ZNkCsf)

<html>

<head>

</head>

<body>

<p **style = "color:red;"**>

This text will be written in red.

</p>

</body>

</html>

It will produce the following result −

## Set the Text Direction

The following example demonstrates how to set the direction of a text. Possible values are *ltr or rtl*.

[Live Demo](http://tpcg.io/aFDTZb)

<html>

<head>

</head>

<body>

<p **style = "direction:rtl;"**>

This text will be rendered from right to left

</p>

</body>

</html>

It will produce the following result −

## Set the Space between Characters

The following example demonstrates how to set the space between characters. Possible values are *normal or a number specifying space.*.

[Live Demo](http://tpcg.io/xr88Ns)

<html>

<head>

</head>

<body>

<p **style = "letter-spacing:5px;"**>

This text is having space between letters.

</p>

</body>

</html>

It will produce the following result −

## Set the Space between Words

The following example demonstrates how to set the space between words. Possible values are *normal or a number specifying space*.

[Live Demo](http://tpcg.io/HNV9nc)

<html>

<head>

</head>

<body>

<p **style = "word-spacing:5px;"**>

This text is having space between words.

</p>

</body>

</html>

This will produce following result −

## Set the Text Indent

The following example demonstrates how to indent the first line of a paragraph. Possible values are *% or a number specifying indent space*.

[Live Demo](http://tpcg.io/9UDJ5g)

<html>

<head>

</head>

<body>

<p **style = "text-indent:1cm;"**>

This text will have first line indented by 1cm and this line will remain at

its actual position this is done by CSS text-indent property.

</p>

</body>

</html>

It will produce the following result −

## Set the Text Alignment

The following example demonstrates how to align a text. Possible values are *left, right, center, justify*.

[Live Demo](http://tpcg.io/hn0sUv)

<html>

<head>

</head>

<body>

<p **style = "text-align:right;"**>

This will be right aligned.

</p>

<p **style = "text-align:center;"**>

This will be center aligned.

</p>

<p **style = "text-align:left;"**>

This will be left aligned.

</p>

</body>

</html>

This will produce following result −

## Decorating the Text

The following example demonstrates how to decorate a text. Possible values are *none, underline, overline, line-through, blink*.

[Live Demo](http://tpcg.io/2GOkPG)

<html>

<head>

</head>

<body>

<p **style = "text-decoration:underline;"**>

This will be underlined

</p>

<p **style = "text-decoration:line-through;"**>

This will be striked through.

</p>

<p **style = "text-decoration:overline;"**>

This will have a over line.

</p>

<p **style = "text-decoration:blink;"**>

This text will have blinking effect

</p>

</body>

</html>

This will produce following result −

## Set the Text Cases

The following example demonstrates how to set the cases for a text. Possible values are *none, capitalize, uppercase, lowercase*.

[Live Demo](http://tpcg.io/saoobs)

<html>

<head>

</head>

<body>

<p **style = "text-transform:capitalize;"**>

This will be capitalized

</p>

<p **style = "text-transform:uppercase;"**>

This will be in uppercase

</p>

<p **style = "text-transform:lowercase;"**>

This will be in lowercase

</p>

</body>

</html>

This will produce following result −

## Set the White Space between Text

The following example demonstrates how white space inside an element is handled. Possible values are *normal, pre, nowrap*.

[Live Demo](http://tpcg.io/bGsQfo)

<html>

<head>

</head>

<body>

<p **style = "white-space:pre;"**>

This text has a line break and the white-space pre setting

tells the browser to honor it just like the HTML pre tag.

</p>

</body>

</html>

This will produce following result −

## Set the Text Shadow

The following example demonstrates how to set the shadow around a text. This may not be supported by all the browsers.

[Live Demo](http://tpcg.io/pYePl1)

<html>

<head>

</head>

<body>

<p **style = "text-shadow:4px 4px 8px blue;"**>

If your browser supports the CSS text-shadow property,

this text will have a blue shadow.

</p>

</body>

</html>

It will produce the following result −

## CSS Border Properties

The CSS border properties allow you to specify the style, width, and color of an element's border.

I have borders on all sides.

I have a red bottom border.

I have rounded borders.

I have a blue left border.

## CSS Border Style

The border-style property specifies what kind of border to display.

The following values are allowed:

* dotted - Defines a dotted border
* dashed - Defines a dashed border
* solid - Defines a solid border
* double - Defines a double border
* groove - Defines a 3D grooved border. The effect depends on the border-color value
* ridge - Defines a 3D ridged border. The effect depends on the border-color value
* inset - Defines a 3D inset border. The effect depends on the border-color value
* outset - Defines a 3D outset border. The effect depends on the border-color value
* none - Defines no border
* hidden - Defines a hidden border

The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border).

### Example

Demonstration of the different border styles:

p.dotted {border-style: dotted;}  
p.dashed {border-style: dashed;}  
p.solid {border-style: solid;}  
p.double {border-style: double;}  
p.groove {border-style: groove;}  
p.ridge {border-style: ridge;}  
p.inset {border-style: inset;}  
p.outset {border-style: outset;}  
p.none {border-style: none;}  
p.hidden {border-style: hidden;}  
p.mix {border-style: dotted dashed solid double;}

Result:

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

A hidden border.

A mixed border.

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_border-style)

**Note:** None of the OTHER CSS border properties described below will have ANY effect unless the border-style property is set!

## CSS Border Width

The border-width property specifies the width of the four borders.

The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick.

The border-width property can have from one to four values (for the top border, right border, bottom border, and the left border).

5px border-width

### Example

p.one {  
  border-style: solid;  
  border-width: 5px;  
}  
  
p.two {  
  border-style: solid;  
  border-width: medium;  
}  
  
p.three {  
  border-style: solid;  
  border-width: 2px 10px 4px 20px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_border-width)

## CSS Border Color

The border-color property is used to set the color of the four borders.

The color can be set by:

* name - specify a color name, like "red"
* Hex - specify a hex value, like "#ff0000"
* RGB - specify a RGB value, like "rgb(255,0,0)"
* transparent

The border-color property can have from one to four values (for the top border, right border, bottom border, and the left border).

If border-color is not set, it inherits the color of the element.

Red border

### Example

p.one {  
  border-style: solid;  
  border-color: red;  
}  
  
p.two {  
  border-style: solid;  
  border-color: green;  
}  
  
p.three {  
  border-style: solid;  
  border-color: red green blue yellow;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_border-color1)

## CSS Border - Individual Sides

From the examples above you have seen that it is possible to specify a different border for each side.

In CSS, there are also properties for specifying each of the borders (top, right, bottom, and left):

Different Border Styles

### Example

p {  
  border-top-style: dotted;  
  border-right-style: solid;  
  border-bottom-style: dotted;  
  border-left-style: solid;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_border-side)

The example above gives the same result as this:

### Example

p {  
  border-style: dotted solid;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_border-side2)

So, here is how it works:

If the border-style property has four values:

* **border-style: dotted solid double dashed;**
  + top border is dotted
  + right border is solid
  + bottom border is double
  + left border is dashed

If the border-style property has three values:

* **border-style: dotted solid double;**
  + top border is dotted
  + right and left borders are solid
  + bottom border is double

If the border-style property has two values:

* **border-style: dotted solid;**
  + top and bottom borders are dotted
  + right and left borders are solid

If the border-style property has one value:

* **border-style: dotted;**
  + all four borders are dotted

The border-style property is used in the example above. However, it also works with border-width and border-color.

## CSS Border - Shorthand Property

As you can see from the examples above, there are many properties to consider when dealing with borders.

To shorten the code, it is also possible to specify all the individual border properties in one property.

The border property is a shorthand property for the following individual border properties:

* border-width
* border-style (required)
* border-color

### Example

p {  
  border: 5px solid red;  
}

Result:

Some text

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_border)

You can also specify all the individual border properties for just one side:

### Left Border

p {  
  border-left: 6px solid red;  
  background-color: lightgrey;  
}

Result:

Some text

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_border_left)

### Bottom Border

p {  
  border-bottom: 6px solid red;  
  background-color: lightgrey;  
}

Result:

Some text

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_border_bottom)

## CSS Rounded Borders

The border-radius property is used to add rounded borders to an element:

Normal border

Round border

Rounder border

Roundest border

### Example

p {  
  border: 2px solid red;  
  border-radius: 5px;  
}

## CSS Margins

The CSS margin properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

## Margin - Individual Sides

CSS has properties for specifying the margin for each side of an element:

* margin-top
* margin-right
* margin-bottom
* margin-left

All the margin properties can have the following values:

* auto - the browser calculates the margin
* length - specifies a margin in px, pt, cm, etc.
* % - specifies a margin in % of the width of the containing element
* inherit - specifies that the margin should be inherited from the parent element

**Tip:** Negative values are allowed.

### Example

Set different margins for all four sides of a <p> element:

p {  
  margin-top: 100px;  
  margin-bottom: 100px;  
  margin-right: 150px;  
  margin-left: 80px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_sides)

## Margin - Shorthand Property

To shorten the code, it is possible to specify all the margin properties in one property.

The margin property is a shorthand property for the following individual margin properties:

* margin-top
* margin-right
* margin-bottom
* margin-left

So, here is how it works:

If the margin property has four values:

* **margin: 25px 50px 75px 100px;**
  + top margin is 25px
  + right margin is 50px
  + bottom margin is 75px
  + left margin is 100px

### Example

Use the margin shorthand property with four values:

p {  
  margin: 25px 50px 75px 100px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_shorthand_4val)

If the margin property has three values:

* **margin: 25px 50px 75px;**
  + top margin is 25px
  + right and left margins are 50px
  + bottom margin is 75px

### Example

Use the margin shorthand property with three values:

p {  
  margin: 25px 50px 75px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_shorthand_3val)

If the margin property has two values:

* **margin: 25px 50px;**
  + top and bottom margins are 25px
  + right and left margins are 50px

### Example

Use the margin shorthand property with two values:

p {  
  margin: 25px 50px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_shorthand_2val)

If the margin property has one value:

* **margin: 25px;**
  + all four margins are 25px

### Example

Use the margin shorthand property with one value:

p {  
  margin: 25px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_shorthand_1val)

## The auto Value

You can set the margin property to auto to horizontally center the element within its container.

The element will then take up the specified width, and the remaining space will be split equally between the left and right margins.

### Example

Use margin: auto:

div {  
  width: 300px;  
  margin: auto;  
  border: 1px solid red;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_auto)

## The inherit Value

This example lets the left margin of the <p class="ex1"> element be inherited from the parent element (<div>):

### Example

Use of the inherit value:

div {  
  border: 1px solid red;  
  margin-left: 100px;  
}  
  
p.ex1 {  
  margin-left: inherit;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_margin-left_inherit)

## Margin Collapse

Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.

This does not happen on left and right margins! Only top and bottom margins!

Look at the following example:

### Example

Demonstration of margin collapse:

h1 {  
  margin: 0 0 50px 0;  
}  
  
h2 {  
  margin: 20px 0 0 0;  
}

## CSS Padding

The CSS padding properties are used to generate space around an element's content, inside of any defined borders.

With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

## Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

* padding-top
* padding-right
* padding-bottom
* padding-left

All the padding properties can have the following values:

* length - specifies a padding in px, pt, cm, etc.
* % - specifies a padding in % of the width of the containing element
* inherit - specifies that the padding should be inherited from the parent element

**Note:** Negative values are not allowed.

### Example

Set different padding for all four sides of a <div> element:

div {  
  padding-top: 50px;  
  padding-right: 30px;  
  padding-bottom: 50px;  
  padding-left: 80px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_padding_sides)

## Padding - Shorthand Property

To shorten the code, it is possible to specify all the padding properties in one property.

The padding property is a shorthand property for the following individual padding properties:

* padding-top
* padding-right
* padding-bottom
* padding-left

So, here is how it works:

If the padding property has four values:

* **padding: 25px 50px 75px 100px;**
  + top padding is 25px
  + right padding is 50px
  + bottom padding is 75px
  + left padding is 100px

### Example

Use the padding shorthand property with four values:

div {  
  padding: 25px 50px 75px 100px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_padding_shorthand_4val)

If the padding property has three values:

* **padding: 25px 50px 75px;**
  + top padding is 25px
  + right and left paddings are 50px
  + bottom padding is 75px

### Example

Use the padding shorthand property with three values:

div {  
  padding: 25px 50px 75px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_padding_shorthand_3val)

If the padding property has two values:

* **padding: 25px 50px;**
  + top and bottom paddings are 25px
  + right and left paddings are 50px

### Example

Use the padding shorthand property with two values:

div {  
  padding: 25px 50px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_padding_shorthand_2val)

If the padding property has one value:

* **padding: 25px;**
  + all four paddings are 25px

### Example

Use the padding shorthand property with one value:

div {  
  padding: 25px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_padding_shorthand_1val)

## Padding and Element Width

The CSS width property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element ([the box model](https://www.w3schools.com/css/css_boxmodel.asp)).

So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result.

### Example

Here, the <div> element is given a width of 300px. However, the actual width of the <div> element will be 350px (300px + 25px of left padding + 25px of right padding):

div {  
  width: 300px;  
  padding: 25px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_padding_width)

To keep the width at 300px, no matter the amount of padding, you can use the box-sizing property. This causes the element to maintain its width; if you increase the padding, the available content space will decrease.

### Example

Use the box-sizing property to keep the width at 300px, no matter the amount of padding:

div {  
  width: 300px;  
  padding: 25px;  
  box-sizing: border-box;  
}

## Text Color

The color property is used to set the color of the text. The color is specified by:

* a color name - like "red"
* a HEX value - like "#ff0000"
* an RGB value - like "rgb(255,0,0)"

Look at [CSS Color Values](https://www.w3schools.com/cssref/css_colors_legal.asp) for a complete list of possible color values.

The default text color for a page is defined in the body selector.

### Example

body {  
  color: blue;  
}  
  
h1 {  
  color: green;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_color)

**Note:** For W3C compliant CSS: If you define the color property, you must also define the background-color.

## Text Alignment

The text-align property is used to set the horizontal alignment of a text.

A text can be left or right aligned, centered, or justified.

The following example shows center aligned, and left and right aligned text (left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left):

### Example

h1 {  
  text-align: center;  
}  
  
h2 {  
  text-align: left;  
}  
  
h3 {  
  text-align: right;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_text-align)

When the text-align property is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight (like in magazines and newspapers):

### Example

div {  
  text-align: justify;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_text-align_all)

## Text Decoration

The text-decoration property is used to set or remove decorations from text.

The value text-decoration: none; is often used to remove underlines from links:

### Example

a {  
  text-decoration: none;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_text-decoration_link)

The other text-decoration values are used to decorate text:

### Example

h1 {  
  text-decoration: overline;  
}  
  
h2 {  
  text-decoration: line-through;  
}  
  
h3 {  
  text-decoration: underline;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_text-decoration)

**Note:** It is not recommended to underline text that is not a link, as this often confuses the reader.

## Text Transformation

The text-transform property is used to specify uppercase and lowercase letters in a text.

It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word:

### Example

p.uppercase {  
  text-transform: uppercase;  
}  
  
p.lowercase {  
  text-transform: lowercase;  
}  
  
p.capitalize {  
  text-transform: capitalize;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_text-transform)

## Text Indentation

The text-indent property is used to specify the indentation of the first line of a text:

### Example

p {  
  text-indent: 50px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_text-indent)

## Letter Spacing

The letter-spacing property is used to specify the space between the characters in a text.

The following example demonstrates how to increase or decrease the space between characters:

### Example

h1 {  
  letter-spacing: 3px;  
}  
  
h2 {  
  letter-spacing: -3px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_letter-spacing)

## Line Height

The line-height property is used to specify the space between lines:

### Example

p.small {  
  line-height: 0.8;  
}  
  
p.big {  
  line-height: 1.8;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_line-height)

## Text Direction

The direction property is used to change the text direction of an element:

### Example

p {  
  direction: rtl;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_text_direction)

## Word Spacing

The word-spacing property is used to specify the space between the words in a text.

The following example demonstrates how to increase or decrease the space between words:

### Example

h1 {  
  word-spacing: 10px;  
}  
  
h2 {  
  word-spacing: -5px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_text_word-spacing)

## Text Shadow

The text-shadow property adds shadow to text.

The following example specifies the position of the horizontal shadow (3px), the position of the vertical shadow (2px) and the color of the shadow (red):

### Example

h1 {  
  text-shadow: 3px 2px red;  
}

With CSS, links can be styled in different ways.

[Text Link](javascript:void(0)) [Text Link](javascript:void(0)) [Link Button](javascript:void(0)) [Link Button](javascript:void(0))

## Styling Links

Links can be styled with any CSS property (e.g. color, font-family, background, etc.).

### Example

a {  
  color: hotpink;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_link_all)

In addition, links can be styled differently depending on what **state** they are in.

The four links states are:

* a:link - a normal, unvisited link
* a:visited - a link the user has visited
* a:hover - a link when the user mouses over it
* a:active - a link the moment it is clicked

### Example

/\* unvisited link \*/  
a:link {  
  color: red;  
}  
  
/\* visited link \*/  
a:visited {  
  color: green;  
}  
  
/\* mouse over link \*/  
a:hover {  
  color: hotpink;  
}  
  
/\* selected link \*/  
a:active {  
  color: blue;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_link)

When setting the style for several link states, there are some order rules:

* a:hover MUST come after a:link and a:visited
* a:active MUST come after a:hover

## Text Decoration

The text-decoration property is mostly used to remove underlines from links:

### Example

a:link {  
  text-decoration: none;  
}  
  
a:visited {  
  text-decoration: none;  
}  
  
a:hover {  
  text-decoration: underline;  
}  
  
a:active {  
  text-decoration: underline;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_link_decoration)

## Background Color

The background-color property can be used to specify a background color for links:

### Example

a:link {  
  background-color: yellow;  
}  
  
a:visited {  
  background-color: cyan;  
}  
  
a:hover {  
  background-color: lightgreen;  
}  
  
a:active {  
  background-color: hotpink;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_link_background)

## Advanced - Link Buttons

This example demonstrates a more advanced example where we combine several CSS properties to display links as boxes/buttons:

### Example

a:link, a:visited {  
  background-color: #f44336;  
  color: white;  
  padding: 14px 25px;  
  text-align: center;  
  text-decoration: none;  
  display: inline-block;  
}  
  
a:hover, a:active {  
  background-color: red;  
}

The position property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

## The position Property

The position property specifies the type of positioning method used for an element.

There are five different position values:

* static
* relative
* fixed
* absolute
* sticky

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position property is set first. They also work differently depending on the position value.

## position: static;

HTML elements are positioned static by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

This <div> element has position: static;

Here is the CSS that is used:

### Example

div.static {  
  position: static;  
  border: 3px solid #73AD21;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_position_static)

## position: relative;

An element with position: relative; is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

This <div> element has position: relative;

Here is the CSS that is used:

### Example

div.relative {  
  position: relative;  
  left: 30px;  
  border: 3px solid #73AD21;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_position_relative)

## position: fixed;

An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

Notice the fixed element in the lower-right corner of the page. Here is the CSS that is used:

### Example

div.fixed {  
  position: fixed;  
  bottom: 0;  
  right: 0;  
  width: 300px;  
  border: 3px solid #73AD21;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_position_fixed)

This <div> element has position: fixed;

## position: absolute;

An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

**Note:** A "positioned" element is one whose position is anything except static.

Here is a simple example:

This <div> element has position: relative;

This <div> element has position: absolute;

Here is the CSS that is used:

### Example

div.relative {  
  position: relative;  
  width: 400px;  
  height: 200px;  
  border: 3px solid #73AD21;  
}  
  
div.absolute {  
  position: absolute;  
  top: 80px;  
  right: 0;  
  width: 200px;  
  height: 100px;  
  border: 3px solid #73AD21;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_position_absolute)

## position: sticky;

An element with position: sticky; is positioned based on the user's scroll position.

A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

**Note:**Internet Explorer, Edge 15 and earlier versions do not support sticky positioning. Safari requires a -webkit- prefix (see example below). You must also specify at least one of top, right, bottom or left for sticky positioning to work.

In this example, the sticky element sticks to the top of the page (top: 0), when you reach its scroll position.

### Example

div.sticky {  
  position: -webkit-sticky; /\* Safari \*/  
  position: sticky;  
  top: 0;  
  background-color: green;  
  border: 2px solid #4CAF50;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_position_sticky)

## Overlapping Elements

When elements are positioned, they can overlap other elements.

The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).

An element can have a positive or negative stack order:

# This is a heading



Because the image has a z-index of -1, it will be placed behind the text.

### Example

img {  
  position: absolute;  
  left: 0px;  
  top: 0px;  
  z-index: -1;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_zindex)

An element with greater stack order is always in front of an element with a lower stack order.

**Note:** If two positioned elements overlap without a z-index specified, the element positioned last in the HTML code will be shown on top.

## Positioning Text In an Image

How to position text over an image:

### Example



## CSS Overflow

The overflow property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

The overflow property has the following values:

* visible - Default. The overflow is not clipped. The content renders outside the element's box
* hidden - The overflow is clipped, and the rest of the content will be invisible
* scroll - The overflow is clipped, and a scrollbar is added to see the rest of the content
* auto - Similar to scroll, but it adds scrollbars only when necessary

**Note:** The overflow property only works for block elements with a specified height.

**Note:** In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set).

## overflow: visible

By default, the overflow is visible, meaning that it is not clipped and it renders outside the element's box:

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

### Example

div {  
  width: 200px;  
  height: 50px;  
  background-color: #eee;  
  overflow: visible;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_visible)

## overflow: hidden

With the hidden value, the overflow is clipped, and the rest of the content is hidden:

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

### Example

div {  
  overflow: hidden;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_hidden)

## overflow: scroll

Setting the value to scroll, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

### Example

div {  
  overflow: scroll;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_scroll)

## overflow: auto

The auto value is similar to scroll, but it adds scrollbars only when necessary:

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

### Example

div {  
  overflow: auto;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_auto)

## overflow-x and overflow-y

The overflow-x and overflow-y properties specifies whether to change the overflow of content just horizontally or vertically (or both):

overflow-x specifies what to do with the left/right edges of the content.  
overflow-y specifies what to do with the top/bottom edges of the content.

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

### Example

div {  
  overflow-x: hidden; /\* Hide horizontal scrollbar \*/  
  overflow-y: scroll; /\* Add vertical scrollbar \*/  
}

The CSS float property specifies how an element should float.

The CSS clear property specifies what elements can float beside the cleared element and on which side.

## The float Property

The float property is used for positioning and formatting content e.g. let an image float left to the text in a container.

The float property can have one of the following values:

* left - The element floats to the left of its container
* right - The element floats to the right of its container
* none - The element does not float (will be displayed just where it occurs in the text). This is default
* inherit - The element inherits the float value of its parent

In its simplest use, the float property can be used to wrap text around images.

## Example - float: right;

The following example specifies that an image should float to the **right** in a text:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

### Example

img {  
  float: right;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_float)

## Example - float: left;

The following example specifies that an image should float to the **left** in a text:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

### Example

img {  
  float: left;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_float2)

## Example - No float

In the following example the image will be displayed just where it occurs in the text (float: none;):

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

### Example

img {  
  float: none;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_float_none)

## The clear Property

The clear property specifies what elements can float beside the cleared element and on which side.

The clear property can have one of the following values:

* none - Allows floating elements on both sides. This is default
* left - No floating elements allowed on the left side
* right- No floating elements allowed on the right side
* both - No floating elements allowed on either the left or the right side
* inherit - The element inherits the clear value of its parent

The most common way to use the clear property is after you have used a float property on an element.

When clearing floats, you should match the clear to the float: If an element is floated to the left, then you should clear to the left. Your floated element will continue to float, but the cleared element will appear below it on the web page.

The following example clears the float to the left. Means that no floating elements are allowed on the left side (of the div):

### Example

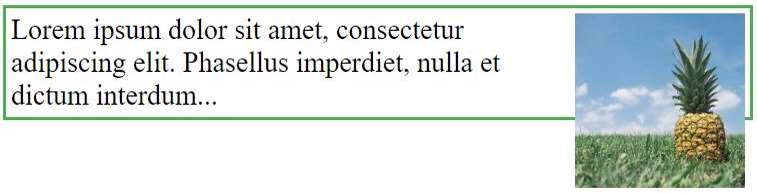
div {  
  clear: left;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_clear)

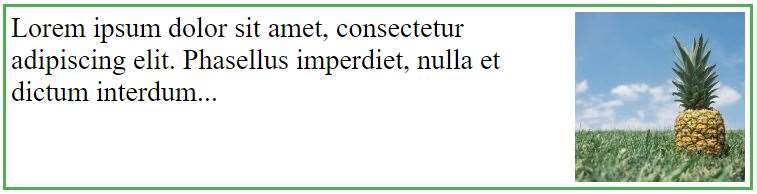
## The clearfix Hack

If an element is taller than the element containing it, and it is floated, it will "overflow" outside of its container:

### Without Clearfix



### With Clearfix



Then we can add overflow: auto; to the containing element to fix this problem:

### Example

.clearfix {  
  overflow: auto;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_clearfix)

The overflow: auto clearfix works well as long as you are able to keep control of your margins and padding (else you might see scrollbars). The **new, modern clearfix hack** however, is safer to use, and the following code is used for most webpages:

### Example

.clearfix::after {  
  content: "";  
  clear: both;  
  display: table;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_clearfix2)

You will learn more about the ::after pseudo-element in a later chapter.

## Grid of Boxes / Equal Width Boxes

Box 1

Box 2

Box 1

Box 2

Box 3

With the float property, it is easy to float boxes of content side by side:

### Example

\* {  
  box-sizing: border-box;  
}  
  
.box {  
  float: left;  
  width: 33.33%; /\* three boxes (use 25% for four, and 50% for two, etc) \*/  
  padding: 50px; /\* if you want space between the images \*/  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_float_boxes)

**What is box-sizing?**

You can easily create three floating boxes side by side. However, when you add something that enlarges the width of each box (e.g. padding or borders), the box will break. The box-sizing property allows us to include the padding and border in the box's total width (and height), making sure that the padding stays inside of the box and that it does not break.

You can read more about the box-sizing property in our [CSS Box Sizing Chapter](https://www.w3schools.com/css/css3_box-sizing.asp).

## Images Side By Side







The grid of boxes can also be used to display images side by side:

### Example

.img-container {  
  float: left;  
  width: 33.33%; /\* three containers (use 25% for four, and 50% for two, etc) \*/  
  padding: 5px; /\* if you want space between the images \*/  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_float_images_side)

## Equal Height Boxes

In the previous example, you learned how to float boxes side by side with an equal width. However, it is not easy to create floating boxes with equal heights. A quick fix however, is to set a fixed height, like in the example below:

## Box 1

Some content, some content, some content

## Box 2

Some content, some content, some content

Some content, some content, some content

Some content, some content, some content

### Example

.box {  
  height: 500px;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_float_boxes_height)

**However**, this is not very flexible. It is ok if you can guarantee that the boxes will always have the same amount of content in them. But many times, the content is not the same. If you try the example above on a mobile phone, you will see that the second box's content will be displayed outside of the box. This is where CSS3 Flexbox comes in handy - as it can automatically stretch boxes to be as long as the longest box:

### Example

Using **Flexbox** to create flexible boxes:

Box 1 - This is some text to make sure that the content gets really tall. This is some text to make sure that the content gets really tall. This is some text to make sure that the content gets really tall.

Box 2 - My height will follow Box 1.

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_float_boxes_flex)

The only problem with Flexbox is that it does not work in Internet Explorer 10 or earlier versions. You can read more about the Flexbox Layout Module in our [CSS Flexbox Chapter](https://www.w3schools.com/css/css3_flexbox.asp).

## Navigation Menu

Use float with a list of hyperlinks to create a horizontal menu:

### Example

* [Home](javascript:void(0))
* [News](javascript:void(0))
* [Contact](javascript:void(0))
* [About](javascript:void(0))

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss_float5)

## Web Layout Example

It is also common to do entire web layouts using the float property:

### Example

.header, .footer {  
  background-color: grey;  
  color: white;  
  padding: 15px;  
}  
  
.column {  
  float: left;  
  padding: 15px;  
}  
  
.clearfix::after {  
  content: "";  
  clear: both;  
  display: table;  
}  
  
.menu {  
  width: 25%;  
}  
  
.content {  
  width: 75%;  
}

**JavaScript** is a lightweight, interpreted **programming** language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. **JavaScript** is very easy to implement because it is integrated with HTML. It is open and cross-platform.

## Why to Learn Javascript

**Javascript** is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Javascript:

* Javascript is the most popular **programming language** in the world and that makes it a programmer’s great choice. Once you learnt Javascript, it helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS etc.
* Javascript is everywhere, it comes installed on every modern web browser and so to learn Javascript you really do not need any special environment setup. For example Chrome, Mozilla Firefox , Safari and every browser you know as of today, supports Javascript.
* Javascript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.
* JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as Javascript Programmer.
* Due to high demand, there is tons of job growth and high pay for those who know JavaScript. You can navigate over to different job sites to see what having JavaScript skills looks like in the job market.
* Great thing about Javascript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.

There could be 1000s of good reasons to learn Javascript Programming. But one thing for sure, to learn any **programming language**, not only Javascript, you just need to code, and code and finally code until you become expert.

## Hello World using Javascript

Just to give you a little excitement about **Javascript programming**, I'm going to give you a small conventional Javascript Hello World program, You can try it using Demo link

[Live Demo](http://tpcg.io/aMKLYm)

<html>

<body>

<script language = "javascript" type = "text/javascript">

<!--

document.write("Hello World!")

//-->

</script>

</body>

</html>

There are many useful **Javascript frameworks** and libraries available:

* Angular
* React
* jQuery
* Vue.js
* Ext.js
* Ember.js
* Meteor
* Mithril
* Node.js
* Polymer
* Aurelia
* Backbone.js

It is really impossible to give a complete list of all the available Javascript frameworks and libraries. The Javascript world is just too large and too much new is happening.

## Applications of Javascript Programming

As mentioned before, **Javascript** is one of the most widely used **programming languages** (Front-end as well as Back-end). It has it's presence in almost every area of software development. I'm going to list few of them here:

* **Client side validation** - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validting those inputs at front-end itself.
* **Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.
* **User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
* **Back-end Data Loading** - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
* **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
* **Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

## What is JavaScript ?

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as **LiveScript,** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The [ECMA-262 Specification](http://www.ecma-international.org/publications/index.html) defined a standard version of the core JavaScript language.

* JavaScript is a lightweight, interpreted programming language.
* Designed for creating network-centric applications.
* Complementary to and integrated with Java.
* Complementary to and integrated with HTML.
* Open and cross-platform

## Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

## Advantages of JavaScript

The merits of using JavaScript are −

* **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
* **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
* **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
* **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

## Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

* Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
* JavaScript cannot be used for networking applications because there is no such support available.
* JavaScript doesn't have any multi-threading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

## JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here −

* **Microsoft FrontPage** − Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
* **Macromedia Dreamweaver MX** − Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
* **Macromedia HomeSite 5** − HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

## Where is JavaScript Today ?

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.

The specification for JavaScript 2.0 can be found on the following site: <http://www.ecmascript.org/>

Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both the languages still support the features that are not a part of the standard.

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

<script ...>

JavaScript code

</script>

The script tag takes two important attributes −

* **Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
* **Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like −

<script language = "javascript" type = "text/javascript">

JavaScript code

</script>

## Your First JavaScript Code

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

[Live Demo](http://tpcg.io/aMKLYm)

<html>

<body>

<script language = "javascript" type = "text/javascript">

<!--

document.write("Hello World!")

//-->

</script>

</body>

</html>

This code will produce the following result −

Hello World!

## Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

## Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

<script language = "javascript" type = "text/javascript">

<!--

var1 = 10

var2 = 20

//-->

</script>

But when formatted in a single line as follows, you must use semicolons −

<script language = "javascript" type = "text/javascript">

<!--

var1 = 10; var2 = 20;

//-->

</script>

**Note** − It is a good programming practice to use semicolons.

## Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

**NOTE** − Care should be taken while writing variable and function names in JavaScript.

## Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus −

* Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
* Any text between the characters /\* and \*/ is treated as a comment. This may span multiple lines.
* JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
* The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

### Example

The following example shows how to use comments in JavaScript.

<script language = "javascript" type = "text/javascript">

<!--

// This is a comment. It is similar to comments in C++

/\*

\* This is a multi-line comment in JavaScript

\* It is very similar to comments in C Programming

\*/

//-->

</script>

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows −

* Script in <head>...</head> section.
* Script in <body>...</body> section.
* Script in <body>...</body> and <head>...</head> sections.
* Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

## JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows −

[Live Demo](http://tpcg.io/S11ZZW)

<html>

<head>

<script type = "text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<input type = "button" onclick = "sayHello()" value = "Say Hello" />

</body>

</html>

This code will produce the following results −

## JavaScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

[Live Demo](http://tpcg.io/fjLLaF)

<html>

<head>

</head>

<body>

<script type = "text/javascript">

<!--

document.write("Hello World")

//-->

</script>

<p>This is web page body </p>

</body>

</html>

This code will produce the following results −

## JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows −

[Live Demo](http://tpcg.io/MqrfLr)

<html>

<head>

<script type = "text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<script type = "text/javascript">

<!--

document.write("Hello World")

//-->

</script>

<input type = "button" onclick = "sayHello()" value = "Say Hello" />

</body>

</html>

This code will produce the following result −

## JavaScript in External File

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using **script** tag and its **src** attribute.

<html>

<head>

<script type = "text/javascript" src = "filename.js" ></script>

</head>

<body>

.......

</body>

</html>

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

function sayHello() {

alert("Hello World")

}

## JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types −

* **Numbers,** eg. 123, 120.50 etc.
* **Strings** of text e.g. "This text string" etc.
* **Boolean** e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined,** each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

**Note** − JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

## JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

<script type = "text/javascript">

<!--

var money;

var name;

//-->

</script>

You can also declare multiple variables with the same **var** keyword as follows −

<script type = "text/javascript">

<!--

var money, name;

//-->

</script>

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

<script type = "text/javascript">

<!--

var name = "Ali";

var money;

money = 2000.50;

//-->

</script>

**Note** − Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

## JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

* **Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
* **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

[Live Demo](http://tpcg.io/wAb0EM)

<html>

<body onload = checkscope();>

<script type = "text/javascript">

<!--

var myVar = "global"; // Declare a global variable

function checkscope( ) {

var myVar = "local"; // Declare a local variable

document.write(myVar);

}

//-->

</script>

</body>

</html>

This produces the following result −

local

## JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

* You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
* JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **\_123test** is a valid one.
* JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

## JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | else | instanceof | switch |
| boolean | enum | int | synchronized |
| break | export | interface | this |
| byte | extends | long | throw |
| case | false | native | throws |
| catch | final | new | transient |
| char | finally | null | true |
| class | float | package | try |
| const | for | private | typeof |
| continue | function | protected | var |
| debugger | goto | public | void |
| default | if | return | volatile |
| delete | implements | short | while |
| do | import | static | with |
| double | in | super |  |

## What is an Operator?

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and ‘+’ is called the **operator**. JavaScript supports the following types of operators.

* Arithmetic Operators
* Comparison Operators
* Logical (or Relational) Operators
* Assignment Operators
* Conditional (or ternary) Operators

Lets have a look on all operators one by one.

## Arithmetic Operators

JavaScript supports the following arithmetic operators −

Assume variable A holds 10 and variable B holds 20, then −

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **+ (Addition)**  Adds two operands  **Ex:** A + B will give 30 |
| 2 | **- (Subtraction)**  Subtracts the second operand from the first  **Ex:** A - B will give -10 |
| 3 | **\* (Multiplication)**  Multiply both operands  **Ex:** A \* B will give 200 |
| 4 | **/ (Division)**  Divide the numerator by the denominator  **Ex:** B / A will give 2 |
| 5 | **% (Modulus)**  Outputs the remainder of an integer division  **Ex:** B % A will give 0 |
| 6 | **++ (Increment)**  Increases an integer value by one  **Ex:** A++ will give 11 |
| 7 | **-- (Decrement)**  Decreases an integer value by one  **Ex:** A-- will give 9 |

**Note** − Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

### Example

The following code shows how to use arithmetic operators in JavaScript.

[Live Demo](http://tpcg.io/lnSWZs)

<html>

<body>

<script type = "text/javascript">

<!--

var a = 33;

var b = 10;

var c = "Test";

var linebreak = "<br />";

document.write("a + b = ");

result = a + b;

document.write(result);

document.write(linebreak);

document.write("a - b = ");

result = a - b;

document.write(result);

document.write(linebreak);

document.write("a / b = ");

result = a / b;

document.write(result);

document.write(linebreak);

document.write("a % b = ");

result = a % b;

document.write(result);

document.write(linebreak);

document.write("a + b + c = ");

result = a + b + c;

document.write(result);

document.write(linebreak);

a = ++a;

document.write("++a = ");

result = ++a;

document.write(result);

document.write(linebreak);

b = --b;

document.write("--b = ");

result = --b;

document.write(result);

document.write(linebreak);

//-->

</script>

Set the variables to different values and then try...

</body>

</html>

### Output

a + b = 43

a - b = 23

a / b = 3.3

a % b = 3

a + b + c = 43Test

++a = 35

--b = 8

Set the variables to different values and then try...

## Comparison Operators

JavaScript supports the following comparison operators −

Assume variable A holds 10 and variable B holds 20, then −

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **= = (Equal)**  Checks if the value of two operands are equal or not, if yes, then the condition becomes true.  **Ex:** (A == B) is not true. |
| 2 | **!= (Not Equal)**  Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.  **Ex:** (A != B) is true. |
| 3 | **> (Greater than)**  Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.  **Ex:** (A > B) is not true. |
| 4 | **< (Less than)**  Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.  **Ex:** (A < B) is true. |
| 5 | **>= (Greater than or Equal to)**  Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.  **Ex:** (A >= B) is not true. |
| 6 | **<= (Less than or Equal to)**  Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.  **Ex:** (A <= B) is true. |

### Example

The following code shows how to use comparison operators in JavaScript.

[Live Demo](http://tpcg.io/vEhWxM)

<html>

<body>

<script type = "text/javascript">

<!--

var a = 10;

var b = 20;

var linebreak = "<br />";

document.write("(a == b) => ");

result = (a == b);

document.write(result);

document.write(linebreak);

document.write("(a < b) => ");

result = (a < b);

document.write(result);

document.write(linebreak);

document.write("(a > b) => ");

result = (a > b);

document.write(result);

document.write(linebreak);

document.write("(a != b) => ");

result = (a != b);

document.write(result);

document.write(linebreak);

document.write("(a >= b) => ");

result = (a >= b);

document.write(result);

document.write(linebreak);

document.write("(a <= b) => ");

result = (a <= b);

document.write(result);

document.write(linebreak);

//-->

</script>

Set the variables to different values and different operators and then try...

</body>

</html>

### Output

(a == b) => false

(a < b) => true

(a > b) => false

(a != b) => true

(a >= b) => false

a <= b) => true

Set the variables to different values and different operators and then try...

## Logical Operators

JavaScript supports the following logical operators −

Assume variable A holds 10 and variable B holds 20, then −

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **&& (Logical AND)**  If both the operands are non-zero, then the condition becomes true.  **Ex:** (A && B) is true. |
| 2 | **|| (Logical OR)**  If any of the two operands are non-zero, then the condition becomes true.  **Ex:** (A || B) is true. |
| 3 | **! (Logical NOT)**  Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.  **Ex:** ! (A && B) is false. |

### Example

Try the following code to learn how to implement Logical Operators in JavaScript.

[Live Demo](http://tpcg.io/oiAvz6)

<html>

<body>

<script type = "text/javascript">

<!--

var a = true;

var b = false;

var linebreak = "<br />";

document.write("(a && b) => ");

result = (a && b);

document.write(result);

document.write(linebreak);

document.write("(a || b) => ");

result = (a || b);

document.write(result);

document.write(linebreak);

document.write("!(a && b) => ");

result = (!(a && b));

document.write(result);

document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

### Output

(a && b) => false

(a || b) => true

!(a && b) => true

Set the variables to different values and different operators and then try...

## Bitwise Operators

JavaScript supports the following bitwise operators −

Assume variable A holds 2 and variable B holds 3, then −

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **& (Bitwise AND)**  It performs a Boolean AND operation on each bit of its integer arguments.  **Ex:** (A & B) is 2. |
| 2 | **| (BitWise OR)**  It performs a Boolean OR operation on each bit of its integer arguments.  **Ex:** (A | B) is 3. |
| 3 | **^ (Bitwise XOR)**  It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.  **Ex:** (A ^ B) is 1. |
| 4 | **~ (Bitwise Not)**  It is a unary operator and operates by reversing all the bits in the operand.  **Ex:** (~B) is -4. |
| 5 | **<< (Left Shift)**  It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.  **Ex:** (A << 1) is 4. |
| 6 | **>> (Right Shift)**  Binary Right Shift Operator. The left operand’s value is moved right by the number of bits specified by the right operand.  **Ex:** (A >> 1) is 1. |
| 7 | **>>> (Right shift with Zero)**  This operator is just like the >> operator, except that the bits shifted in on the left are always zero.  **Ex:** (A >>> 1) is 1. |

### Example

Try the following code to implement Bitwise operator in JavaScript.

[Live Demo](http://tpcg.io/1auWI8)

<html>

<body>

<script type = "text/javascript">

<!--

var a = 2; // Bit presentation 10

var b = 3; // Bit presentation 11

var linebreak = "<br />";

document.write("(a & b) => ");

result = (a & b);

document.write(result);

document.write(linebreak);

document.write("(a | b) => ");

result = (a | b);

document.write(result);

document.write(linebreak);

document.write("(a ^ b) => ");

result = (a ^ b);

document.write(result);

document.write(linebreak);

document.write("(~b) => ");

result = (~b);

document.write(result);

document.write(linebreak);

document.write("(a << b) => ");

result = (a << b);

document.write(result);

document.write(linebreak);

document.write("(a >> b) => ");

result = (a >> b);

document.write(result);

document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

(a & b) => 2

(a | b) => 3

(a ^ b) => 1

(~b) => -4

(a << b) => 16

(a >> b) => 0

Set the variables to different values and different operators and then try...

## Assignment Operators

JavaScript supports the following assignment operators −

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **= (Simple Assignment )**  Assigns values from the right side operand to the left side operand  **Ex:** C = A + B will assign the value of A + B into C |
| 2 | **+= (Add and Assignment)**  It adds the right operand to the left operand and assigns the result to the left operand.  **Ex:** C += A is equivalent to C = C + A |
| 3 | **−= (Subtract and Assignment)**  It subtracts the right operand from the left operand and assigns the result to the left operand.  **Ex:** C -= A is equivalent to C = C - A |
| 4 | **\*= (Multiply and Assignment)**  It multiplies the right operand with the left operand and assigns the result to the left operand.  **Ex:** C \*= A is equivalent to C = C \* A |
| 5 | **/= (Divide and Assignment)**  It divides the left operand with the right operand and assigns the result to the left operand.  **Ex:** C /= A is equivalent to C = C / A |
| 6 | **%= (Modules and Assignment)**  It takes modulus using two operands and assigns the result to the left operand.  **Ex:** C %= A is equivalent to C = C % A |

**Note** − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

### Example

Try the following code to implement assignment operator in JavaScript.

[Live Demo](http://tpcg.io/tmlXco)

<html>

<body>

<script type = "text/javascript">

<!--

var a = 33;

var b = 10;

var linebreak = "<br />";

document.write("Value of a => (a = b) => ");

result = (a = b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a += b) => ");

result = (a += b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a -= b) => ");

result = (a -= b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a \*= b) => ");

result = (a \*= b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a /= b) => ");

result = (a /= b);

document.write(result);

document.write(linebreak);

document.write("Value of a => (a %= b) => ");

result = (a %= b);

document.write(result);

document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

### Output

Value of a => (a = b) => 10

Value of a => (a += b) => 20

Value of a => (a -= b) => 10

Value of a => (a \*= b) => 100

Value of a => (a /= b) => 10

Value of a => (a %= b) => 0

Set the variables to different values and different operators and then try...

## Miscellaneous Operator

We will discuss two operators here that are quite useful in JavaScript: the **conditional operator** (? :) and the **typeof operator**.

### Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

|  |  |
| --- | --- |
| **Sr.No.** | **Operator and Description** |
| 1 | **? : (Conditional )**  If Condition is true? Then value X : Otherwise value Y |

### Example

Try the following code to understand how the Conditional Operator works in JavaScript.

[Live Demo](http://tpcg.io/nqssTK)

<html>

<body>

<script type = "text/javascript">

<!--

var a = 10;

var b = 20;

var linebreak = "<br />";

document.write ("((a > b) ? 100 : 200) => ");

result = (a > b) ? 100 : 200;

document.write(result);

document.write(linebreak);

document.write ("((a < b) ? 100 : 200) => ");

result = (a < b) ? 100 : 200;

document.write(result);

document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

### Output

((a > b) ? 100 : 200) => 200

((a < b) ? 100 : 200) => 100

Set the variables to different values and different operators and then try...

## typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

|  |  |
| --- | --- |
| **Type** | **String Returned by typeof** |
| Number | "number" |
| String | "string" |
| Boolean | "boolean" |
| Object | "object" |
| Function | "function" |
| Undefined | "undefined" |
| Null | "object" |

### Example

The following code shows how to implement **typeof** operator.

[Live Demo](http://tpcg.io/oKWzuc)

<html>

<body>

<script type = "text/javascript">

<!--

var a = 10;

var b = "String";

var linebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric");

document.write("Result => ");

document.write(result);

document.write(linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric");

document.write("Result => ");

document.write(result);

document.write(linebreak);

//-->

</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

### Output

Result => B is String

Result => A is Numeric

Set the variables to different values and different operators and then try...

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the **if..else** statement.

## Flow Chart of if-else

The following flow chart shows how the if-else statement works.



JavaScript supports the following forms of **if..else** statement −

* if statement
* if...else statement
* if...else if... statement.

## if statement

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

### Syntax

The syntax for a basic if statement is as follows −

if (expression) {

Statement(s) to be executed if expression is true

}

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

### Example

Try the following example to understand how the **if** statement works.

[Live Demo](http://tpcg.io/phmRNV)

<html>

<body>

<script type = "text/javascript">

<!--

var age = 20;

if( age > 18 ) {

document.write("<b>Qualifies for driving</b>");

}

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

**Qualifies for driving**

Set the variable to different value and then try...

## if...else statement

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

### Syntax

if (expression) {

Statement(s) to be executed if expression is true

} else {

Statement(s) to be executed if expression is false

}

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the ‘if’ block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

### Example

Try the following code to learn how to implement an if-else statement in JavaScript.

[Live Demo](http://tpcg.io/B5GzT5)

<html>

<body>

<script type = "text/javascript">

<!--

var age = 15;

if( age > 18 ) {

document.write("<b>Qualifies for driving</b>");

} else {

document.write("<b>Does not qualify for driving</b>");

}

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

**Does not qualify for driving**

Set the variable to different value and then try...

## if...else if... statement

The **if...else if...** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions.

### Syntax

The syntax of an if-else-if statement is as follows −

if (expression 1) {

Statement(s) to be executed if expression 1 is true

} else if (expression 2) {

Statement(s) to be executed if expression 2 is true

} else if (expression 3) {

Statement(s) to be executed if expression 3 is true

} else {

Statement(s) to be executed if no expression is true

}

There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

### Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

[Live Demo](http://tpcg.io/NTcFyX)

<html>

<body>

<script type = "text/javascript">

<!--

var book = "maths";

if( book == "history" ) {

document.write("<b>History Book</b>");

} else if( book == "maths" ) {

document.write("<b>Maths Book</b>");

} else if( book == "economics" ) {

document.write("<b>Economics Book</b>");

} else {

document.write("<b>Unknown Book</b>");

}

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

<html>

### Output

**Maths Book**

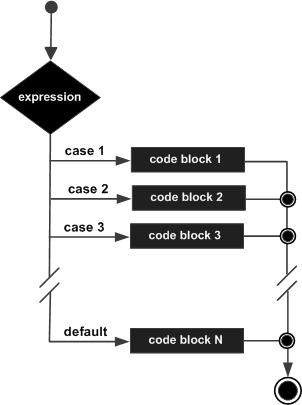
Set the variable to different value and then try...

You can use multiple **if...else…if** statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated **if...else if** statements.

## Flow Chart

The following flow chart explains a switch-case statement works.



### Syntax

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

switch (expression) {

case condition 1: statement(s)

break;

case condition 2: statement(s)

break;

...

case condition n: statement(s)

break;

default: statement(s)

}

The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain **break** statement in **Loop Control** chapter.

### Example

Try the following example to implement switch-case statement.

[Live Demo](http://tpcg.io/pNWbNd)

<html>

<body>

<script type = "text/javascript">

<!--

var grade = 'A';

document.write("Entering switch block<br />");

switch (grade) {

case 'A': document.write("Good job<br />");

break;

case 'B': document.write("Pretty good<br />");

break;

case 'C': document.write("Passed<br />");

break;

case 'D': document.write("Not so good<br />");

break;

case 'F': document.write("Failed<br />");

break;

default: document.write("Unknown grade<br />")

}

document.write("Exiting switch block");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

Entering switch block

Good job

Exiting switch block

Set the variable to different value and then try...

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

[Live Demo](http://tpcg.io/ouqbd3)

<html>

<body>

<script type = "text/javascript">

<!--

var grade = 'A';

document.write("Entering switch block<br />");

switch (grade) {

case 'A': document.write("Good job<br />");

case 'B': document.write("Pretty good<br />");

case 'C': document.write("Passed<br />");

case 'D': document.write("Not so good<br />");

case 'F': document.write("Failed<br />");

default: document.write("Unknown grade<br />")

}

document.write("Exiting switch block");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

Entering switch block

Good job

Pretty good

Passed

Not so good

Failed

Unknown grade

Exiting switch block

Set the variable to different value and then try...

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

JavaScript supports all the necessary loops to ease down the pressure of programming.

## The while Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

### Flow Chart

The flow chart of **while loop** looks as follows −



### Syntax

The syntax of **while loop** in JavaScript is as follows −

while (expression) {

Statement(s) to be executed if expression is true

}

### Example

Try the following example to implement while loop.

[Live Demo](http://tpcg.io/lIlHQD)

<html>

<body>

<script type = "text/javascript">

<!--

var count = 0;

document.write("Starting Loop ");

while (count < 10) {

document.write("Current Count : " + count + "<br />");

count++;

}

document.write("Loop stopped!");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Current Count : 5

Current Count : 6

Current Count : 7

Current Count : 8

Current Count : 9

Loop stopped!

Set the variable to different value and then try...

## The do...while Loop

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

### Flow Chart

The flow chart of a **do-while** loop would be as follows −



### Syntax

The syntax for **do-while** loop in JavaScript is as follows −

do {

Statement(s) to be executed;

} while (expression);

**Note** − Don’t miss the semicolon used at the end of the **do...while** loop.

### Example

Try the following example to learn how to implement a **do-while** loop in JavaScript.

[Live Demo](http://tpcg.io/HCE78l)

<html>

<body>

<script type = "text/javascript">

<!--

var count = 0;

document.write("Starting Loop" + "<br />");

do {

document.write("Current Count : " + count + "<br />");

count++;

}

while (count < 5);

document.write ("Loop stopped!");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Loop Stopped!

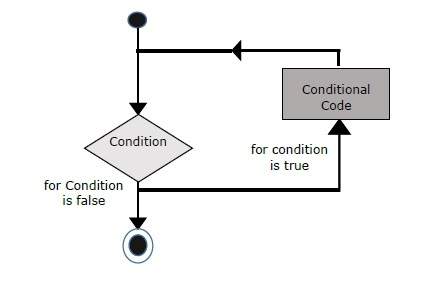
The '**for**' loop is the most compact form of looping. It includes the following three important parts −

* The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
* The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
* The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

## Flow Chart

The flow chart of a **for** loop in JavaScript would be as follows −



### Syntax

The syntax of **for** loop is JavaScript is as follows −

for (initialization; test condition; iteration statement) {

Statement(s) to be executed if test condition is true

}

### Example

Try the following example to learn how a **for** loop works in JavaScript.

[Live Demo](http://tpcg.io/YiJzji)

<html>

<body>

<script type = "text/javascript">

<!--

var count;

document.write("Starting Loop" + "<br />");

for(count = 0; count < 10; count++) {

document.write("Current Count : " + count );

document.write("<br />");

}

document.write("Loop stopped!");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Current Count : 5

Current Count : 6

Current Count : 7

Current Count : 8

Current Count : 9

Loop stopped!

Set the variable to different value and then try...

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

## Syntax

The syntax of ‘for..in’ loop is −

for (variablename in object) {

statement or block to execute

}

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

### Example

Try the following example to implement ‘for-in’ loop. It prints the web browser’s **Navigator** object.

[Live Demo](http://tpcg.io/omAEHY)

<html>

<body>

<script type = "text/javascript">

<!--

var aProperty;

document.write("Navigator Object Properties<br /> ");

for (aProperty in navigator) {

document.write(aProperty);

document.write("<br />");

}

document.write ("Exiting from the loop!");

//-->

</script>

<p>Set the variable to different object and then try...</p>

</body>

</html>

### Output

Navigator Object Properties

serviceWorker

webkitPersistentStorage

webkitTemporaryStorage

geolocation

doNotTrack

onLine

languages

language

userAgent

product

platform

appVersion

appName

appCodeName

hardwareConcurrency

maxTouchPoints

vendorSub

vendor

productSub

cookieEnabled

mimeTypes

plugins

javaEnabled

getStorageUpdates

getGamepads

webkitGetUserMedia

vibrate

getBattery

sendBeacon

registerProtocolHandler

unregisterProtocolHandler

Exiting from the loop!

Set the variable to different object and then try...

JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

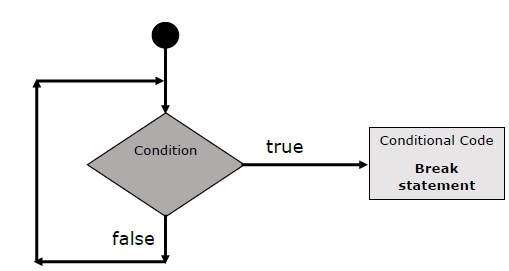
To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

## The break Statement

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

### Flow Chart

The flow chart of a break statement would look as follows −



### Example

The following example illustrates the use of a **break** statement with a while loop. Notice how the loop breaks out early once **x** reaches 5 and reaches to **document.write (..)** statement just below to the closing curly brace −

[Live Demo](http://tpcg.io/Um7xDo)

<html>

<body>

<script type = "text/javascript">

<!--

var x = 1;

document.write("Entering the loop<br /> ");

while (x < 20) {

if (x == 5) {

break; // breaks out of loop completely

}

x = x + 1;

document.write( x + "<br />");

}

document.write("Exiting the loop!<br /> ");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

Entering the loop

2

3

4

5

Exiting the loop!

Set the variable to different value and then try...

We already have seen the usage of **break** statement inside **a switch** statement.

## The continue Statement

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

### Example

This example illustrates the use of a **continue** statement with a while loop. Notice how the **continue** statement is used to skip printing when the index held in variable **x** reaches 5 −

[Live Demo](http://tpcg.io/mBbb2C)

<html>

<body>

<script type = "text/javascript">

<!--

var x = 1;

document.write("Entering the loop<br /> ");

while (x < 10) {

x = x + 1;

if (x == 5) {

continue; // skip rest of the loop body

}

document.write( x + "<br />");

}

document.write("Exiting the loop!<br /> ");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>

### Output

Entering the loop

2

3

4

6

7

8

9

10

Exiting the loop!

Set the variable to different value and then try...

## Using Labels to Control the Flow

Starting from JavaScript 1.2, a label can be used with **break** and **continue** to control the flow more precisely. A **label** is simply an identifier followed by a colon (:) that is applied to a statement or a block of code. We will see two different examples to understand how to use labels with break and continue.

**Note** − Line breaks are not allowed between the **‘continue’** or **‘break’** statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Try the following two examples for a better understanding of Labels.

### Example 1

The following example shows how to implement Label with a break statement.

[Live Demo](http://tpcg.io/voKP9K)

<html>

<body>

<script type = "text/javascript">

<!--

document.write("Entering the loop!<br /> ");

outerloop: // This is the label name

for (var i = 0; i < 5; i++) {

document.write("Outerloop: " + i + "<br />");

innerloop:

for (var j = 0; j < 5; j++) {

if (j > 3 ) break ; // Quit the innermost loop

if (i == 2) break innerloop; // Do the same thing

if (i == 4) break outerloop; // Quit the outer loop

document.write("Innerloop: " + j + " <br />");

}

}

document.write("Exiting the loop!<br /> ");

//-->

</script>

</body>

</html>

### Output

Entering the loop!

Outerloop: 0

Innerloop: 0

Innerloop: 1

Innerloop: 2

Innerloop: 3

Outerloop: 1

Innerloop: 0

Innerloop: 1

Innerloop: 2

Innerloop: 3

Outerloop: 2

Outerloop: 3

Innerloop: 0

Innerloop: 1

Innerloop: 2

Innerloop: 3

Outerloop: 4

Exiting the loop!

### Example 2

[Live Demo](http://tpcg.io/P8njxO)

<html>

<body>

<script type = "text/javascript">

<!--

document.write("Entering the loop!<br /> ");

outerloop: // This is the label name

for (var i = 0; i < 3; i++) {

document.write("Outerloop: " + i + "<br />");

for (var j = 0; j < 5; j++) {

if (j == 3) {

continue outerloop;

}

document.write("Innerloop: " + j + "<br />");

}

}

document.write("Exiting the loop!<br /> ");

//-->

</script>

</body>

</html>

### Output

Entering the loop!

Outerloop: 0

Innerloop: 0

Innerloop: 1

Innerloop: 2

Outerloop: 1

Innerloop: 0

Innerloop: 1

Innerloop: 2

Outerloop: 2

Innerloop: 0

Innerloop: 1

Innerloop: 2

Exiting the loop!

## What is an Event ?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding [HTML Event Reference](https://www.tutorialspoint.com/html/html_events_ref.htm). Here we will see a few examples to understand a relation between Event and JavaScript −

## onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

### Example

Try the following example.

[Live Demo](http://tpcg.io/iyeFLu)

<html>

<head>

<script type = "text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<p>Click the following button and see result</p>

<form>

<input type = "button" onclick = "sayHello()" value = "Say Hello" />

</form>

</body>

</html>

### Output

## onsubmit Event Type

**onsubmit** is an event that occurs when you try to submit a form. You can put your form validation against this event type.

### Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

<html>

<head>

<script type = "text/javascript">

<!--

function validation() {

all validation goes here

.........

return either true or false

}

//-->

</script>

</head>

<body>

<form method = "POST" action = "t.cgi" onsubmit = "return validate()">

.......

<input type = "submit" value = "Submit" />

</form>

</body>

</html>

## onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

[Live Demo](http://tpcg.io/o3Bzlo)

<html>

<head>

<script type = "text/javascript">

<!--

function over() {

document.write ("Mouse Over");

}

function out() {

document.write ("Mouse Out");

}

//-->

</script>

</head>

<body>

<p>Bring your mouse inside the division to see the result:</p>

<div onmouseover = "over()" onmouseout = "out()">

<h2> This is inside the division </h2>

</div>

</body>

</html>

# Arrays

Objects allow you to store keyed collections of values. That’s fine.

But quite often we find that we need an ordered collection, where we have a 1st, a 2nd, a 3rd element and so on. For example, we need that to store a list of something: users, goods, HTML elements etc.

It is not convenient to use an object here, because it provides no methods to manage the order of elements. We can’t insert a new property “between” the existing ones. Objects are just not meant for such use.

There exists a special data structure named Array, to store ordered collections.

## [Declaration](https://javascript.info/array" \l "declaration)

There are two syntaxes for creating an empty array:

let arr = new Array();

let arr = [];

Almost all the time, the second syntax is used. We can supply initial elements in the brackets:

let fruits = ["Apple", "Orange", "Plum"];

Array elements are numbered, starting with zero.

We can get an element by its number in square brackets:

let fruits = ["Apple", "Orange", "Plum"];

alert( fruits[0] ); // Apple

alert( fruits[1] ); // Orange

alert( fruits[2] ); // Plum

We can replace an element:

fruits[2] = 'Pear'; // now ["Apple", "Orange", "Pear"]

…Or add a new one to the array:

fruits[3] = 'Lemon'; // now ["Apple", "Orange", "Pear", "Lemon"]

The total count of the elements in the array is its length:

let fruits = ["Apple", "Orange", "Plum"];

alert( fruits.length ); // 3

We can also use alert to show the whole array.

let fruits = ["Apple", "Orange", "Plum"];

alert( fruits ); // Apple,Orange,Plum

An array can store elements of any type.

For instance:

// mix of values

let arr = [ 'Apple', { name: 'John' }, true, function() { alert('hello'); } ];

// get the object at index 1 and then show its name

alert( arr[1].name ); // John

// get the function at index 3 and run it

arr[3](); // hello

**Trailing comma**

An array, just like an object, may end with a comma:

let fruits = [

"Apple",

"Orange",

"Plum",

];

The “trailing comma” style makes it easier to insert/remove items, because all lines become alike.

## [Methods pop/push, shift/unshift](https://javascript.info/array" \l "methods-pop-push-shift-unshift)

A [queue](https://en.wikipedia.org/wiki/Queue_(abstract_data_type)) is one of the most common uses of an array. In computer science, this means an ordered collection of elements which supports two operations:

* push appends an element to the end.
* shift get an element from the beginning, advancing the queue, so that the 2nd element becomes the 1st.

Arrays support both operations.

In practice we need it very often. For example, a queue of messages that need to be shown on-screen.

There’s another use case for arrays – the data structure named [stack](https://en.wikipedia.org/wiki/Stack_(abstract_data_type)).

It supports two operations:

* push adds an element to the end.
* pop takes an element from the end.

So new elements are added or taken always from the “end”.

A stack is usually illustrated as a pack of cards: new cards are added to the top or taken from the top:

For stacks, the latest pushed item is received first, that’s also called LIFO (Last-In-First-Out) principle. For queues, we have FIFO (First-In-First-Out).

Arrays in JavaScript can work both as a queue and as a stack. They allow you to add/remove elements both to/from the beginning or the end.

In computer science the data structure that allows it is called [deque](https://en.wikipedia.org/wiki/Double-ended_queue).

**Methods that work with the end of the array:**

**pop**

Extracts the last element of the array and returns it:

let fruits = ["Apple", "Orange", "Pear"];

alert( fruits.pop() ); // remove "Pear" and alert it

alert( fruits ); // Apple, Orange

**push**

Append the element to the end of the array:

let fruits = ["Apple", "Orange"];

fruits.push("Pear");

alert( fruits ); // Apple, Orange, Pear

The call fruits.push(...) is equal to fruits[fruits.length] = ....

**Methods that work with the beginning of the array:**

**shift**

Extracts the first element of the array and returns it:

let fruits = ["Apple", "Orange", "Pear"];

alert( fruits.shift() ); // remove Apple and alert it

alert( fruits ); // Orange, Pear

**unshift**

Add the element to the beginning of the array:

let fruits = ["Orange", "Pear"];

fruits.unshift('Apple');

alert( fruits ); // Apple, Orange, Pear

Methods push and unshift can add multiple elements at once:

let fruits = ["Apple"];

fruits.push("Orange", "Peach");

fruits.unshift("Pineapple", "Lemon");

// ["Pineapple", "Lemon", "Apple", "Orange", "Peach"]

alert( fruits );

## [Internals](https://javascript.info/array" \l "internals)

An array is a special kind of object. The square brackets used to access a property arr[0] actually come from the object syntax. That’s essentially the same as obj[key], where arr is the object, while numbers are used as keys.

They extend objects providing special methods to work with ordered collections of data and also the length property. But at the core it’s still an object.

Remember, there are only 7 basic types in JavaScript. Array is an object and thus behaves like an object.

For instance, it is copied by reference:

let fruits = ["Banana"]

let arr = fruits; // copy by reference (two variables reference the same array)

alert( arr === fruits ); // true

arr.push("Pear"); // modify the array by reference

alert( fruits ); // Banana, Pear - 2 items now

…But what makes arrays really special is their internal representation. The engine tries to store its elements in the contiguous memory area, one after another, just as depicted on the illustrations in this chapter, and there are other optimizations as well, to make arrays work really fast.

But they all break if we quit working with an array as with an “ordered collection” and start working with it as if it were a regular object.

For instance, technically we can do this:

let fruits = []; // make an array

fruits[99999] = 5; // assign a property with the index far greater than its length

fruits.age = 25; // create a property with an arbitrary name

That’s possible, because arrays are objects at their base. We can add any properties to them.

But the engine will see that we’re working with the array as with a regular object. Array-specific optimizations are not suited for such cases and will be turned off, their benefits disappear.

The ways to misuse an array:

* Add a non-numeric property like arr.test = 5.
* Make holes, like: add arr[0] and then arr[1000] (and nothing between them).
* Fill the array in the reverse order, like arr[1000], arr[999] and so on.

Please think of arrays as special structures to work with the ordered data. They provide special methods for that. Arrays are carefully tuned inside JavaScript engines to work with contiguous ordered data, please use them this way. And if you need arbitrary keys, chances are high that you actually require a regular object {}.

## [Performance](https://javascript.info/array" \l "performance)

Methods push/pop run fast, while shift/unshift are slow.

Why is it faster to work with the end of an array than with its beginning? Let’s see what happens during the execution:

fruits.shift(); // take 1 element from the start

It’s not enough to take and remove the element with the number 0. Other elements need to be renumbered as well.

The shift operation must do 3 things:

1. Remove the element with the index 0.
2. Move all elements to the left, renumber them from the index 1 to 0, from 2 to 1 and so on.
3. Update the length property.

**The more elements in the array, the more time to move them, more in-memory operations.**

The similar thing happens with unshift: to add an element to the beginning of the array, we need first to move existing elements to the right, increasing their indexes.

And what’s with push/pop? They do not need to move anything. To extract an element from the end, the pop method cleans the index and shortens length.

The actions for the pop operation:

fruits.pop(); // take 1 element from the end

**The pop method does not need to move anything, because other elements keep their indexes. That’s why it’s blazingly fast.**

The similar thing with the push method.

## [Loops](https://javascript.info/array" \l "loops)

One of the oldest ways to cycle array items is the for loop over indexes:

let arr = ["Apple", "Orange", "Pear"];

for (let i = 0; i < arr.length; i++) {

alert( arr[i] );

}

But for arrays there is another form of loop, for..of:

let fruits = ["Apple", "Orange", "Plum"];

// iterates over array elements

for (let fruit of fruits) {

alert( fruit );

}

The for..of doesn’t give access to the number of the current element, just its value, but in most cases that’s enough. And it’s shorter.

Technically, because arrays are objects, it is also possible to use for..in:

let arr = ["Apple", "Orange", "Pear"];

for (let key in arr) {

alert( arr[key] ); // Apple, Orange, Pear

}

But that’s actually a bad idea. There are potential problems with it:

1. The loop for..in iterates over all properties, not only the numeric ones.

There are so-called “array-like” objects in the browser and in other environments, that look like arrays. That is, they have length and indexes properties, but they may also have other non-numeric properties and methods, which we usually don’t need. The for..in loop will list them though. So if we need to work with array-like objects, then these “extra” properties can become a problem.

1. The for..in loop is optimized for generic objects, not arrays, and thus is 10-100 times slower. Of course, it’s still very fast. The speedup may only matter in bottlenecks. But still we should be aware of the difference.

Generally, we shouldn’t use for..in for arrays.

## [A word about “length”](https://javascript.info/array" \l "a-word-about-length)

The length property automatically updates when we modify the array. To be precise, it is actually not the count of values in the array, but the greatest numeric index plus one.

For instance, a single element with a large index gives a big length:

let fruits = [];

fruits[123] = "Apple";

alert( fruits.length ); // 124

Note that we usually don’t use arrays like that.

Another interesting thing about the length property is that it’s writable.

If we increase it manually, nothing interesting happens. But if we decrease it, the array is truncated. The process is irreversible, here’s the example:

let arr = [1, 2, 3, 4, 5];

arr.length = 2; // truncate to 2 elements

alert( arr ); // [1, 2]

arr.length = 5; // return length back

alert( arr[3] ); // undefined: the values do not return

So, the simplest way to clear the array is: arr.length = 0;.

## [new Array()](https://javascript.info/array" \l "new-array)

There is one more syntax to create an array:

let arr = new Array("Apple", "Pear", "etc");

It’s rarely used, because square brackets [] are shorter. Also there’s a tricky feature with it.

If new Array is called with a single argument which is a number, then it creates an array without items, but with the given length.

Let’s see how one can shoot themself in the foot:

let arr = new Array(2); // will it create an array of [2] ?

alert( arr[0] ); // undefined! no elements.

alert( arr.length ); // length 2

In the code above, new Array(number) has all elements undefined.

To evade such surprises, we usually use square brackets, unless we really know what we’re doing.

## [Multidimensional arrays](https://javascript.info/array" \l "multidimensional-arrays)

Arrays can have items that are also arrays. We can use it for multidimensional arrays, for example to store matrices:

let matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

alert( matrix[1][1] ); // 5, the central element

## [toString](https://javascript.info/array" \l "tostring)

Arrays have their own implementation of toString method that returns a comma-separated list of elements.

For instance:

let arr = [1, 2, 3];

alert( arr ); // 1,2,3

alert( String(arr) === '1,2,3' ); // true

Also, let’s try this:

alert( [] + 1 ); // "1"

alert( [1] + 1 ); // "11"

alert( [1,2] + 1 ); // "1,21"

Arrays do not have Symbol.toPrimitive, neither a viable valueOf, they implement only toString conversion, so here [] becomes an empty string, [1] becomes "1" and [1,2] becomes "1,2".

When the binary plus "+" operator adds something to a string, it converts it to a string as well, so the next step looks like this:

alert( "" + 1 ); // "1"

alert( "1" + 1 ); // "11"

alert( "1,2" + 1 ); // "1,21"