# HTML (hypertext markup language)

## HTML Basic

### HTML syntax: tags, elements, attributes

#### HTML Tags

An HTML opening tag has the form <tag-name> (e.g., <p>, <title>), which is associated with a matching closing tag </tag-name> having a leading forward slash, (e.g., </p>, </title>). The tag-name should be in lowercase.

#### Tag's Attributes

Attributes, in the form of name="value" pairs can be included in the opening tag to provide additional information about the element.

**Example 1**: In <html lang="en">, the attribute lang="en" specifies the natural language for this document.

**Example 2**: In <meta charset="utf-8">, the attribute charset="utf-8" specifies the character encoding scheme.

**Example 3**: An <img> (image) tag may contain these attributes:

<img src="logo.gif" alt="logo" width="50" height="30" />

Multiple attributes are separated by space, as follows:

<tag-name attName1="attValue1" attName2="attValue2" ...> ... </tag-name>

#### HTML Elements

An HTML element consists of the opening and closing tags, and the content in between, e.g., <p>A for apple</p>, <strong>Caution!</strong>.

There are two types of elements:

Container Element**:** A container element has an opening tag <tag-name> that activates an effect to its content, and a matching closing tag </tag-name> to discontinue the effect. In other words, container elements apply formatting to their contents. For example:

<h1>The h1 tags enclose a heading level 1</h1>

<p>The p tags is used to <em>markup</em> a <strong>paragraph</strong>.</p>

Standalone Element: A standalone element does not enclose content but is used to achieve a certain effect, e.g., <hr> is used to draw a horizontal rule; <br> to introduce a manual line-break; and <img> for embedding an external image. In XHTML, you need to end a standalone element with a trailing '/' in the opening tag. For examples:

<br />

<hr />

<img src="logo.gif" />

#### Escape sequence for Reserved Characters

HTML uses characters such as <, >, ", & as markup tags' delimiters. Hence, these characters are reversed and cannot be used in the text directly. An escape sequence in the form of &xxx; (begins with & and ends with ; and contain the code xxx) is used for these reserved characters and other special characters.

The five escape sequences that are often used: " (&quot;), < (&lt;), > (&gt;), & (&amp;) and (&nbsp;).

|  |  |
| --- | --- |
| Character | Entity Reference |
| **"** | **&quot;** |
| **<** | **&lt;** |
| **>** | **&gt;** |
| **&** | **&amp;** |
| **non-breaking space** | **&nbsp;** |
| → ⇒ ↔ ⇔ | &rarr; &rArr; &harr; &hArr; |
| ° (degree) | &deg; |
| © ® € ¢ ¥ | &copy; &reg; &euro; &cent; &yen; |
| ˜ | &tilde; |
| × ± ∞ | &times; &plusmn; &infin; |
| π Π σ Σ ω Ω | &pi; &Pi; &sigma; &Sigma; &omega; &Omega; |
| ≥ ≤ ≡ ≈ | &ge; &le; &equiv; &asymp; |
| ⊂ ⊃ ⊆ ⊇ ∈ | &sub; &sup; &sube; &supe; &isin; |

#### HTML ignores extra white spaces and use &nbsp; to reserve them

Blanks, tabs, new-lines and carriage-returns are collectively known as white spaces. "Extra" white spaces are ignored. That is, only the first white space is recognized and displayed.

<p>See how the extra white spaces,

tabs and

line-breaks are ignored by the

browser.</p>

is equivalent to:

<p>See how the extra white spaces, tabs and line-breaks are ignored by the browser </p>

In order to insert multiple whitespaces, one needs to use &nbsp;

<p>This paragraph contains special character &quot; &lt;, &gt; and &amp;

and those &nbsp;&nbsp; words &nbsp;&nbsp; have &nbsp;&nbsp;

more &nbsp;&nbsp; spaces in between.</p>

will become:

This paragraph contains special character " <, > and & and those    words    have    more    spaces in between.

#### HTML Comment <!-- ... -->

HTML comments are enclosed between <!-- and -->. Comments are ignored by the browser.

### Structure of a HTML page

#### A HTML has <html>, <head>, <body>

A html file should have <head> and <body>

<!DOCTYPE html>

<html>

<head>...</head>

<body>...</body>

</html>

The <body>...</body> element defines the BODY section of an HTML document, which encloses the content to be displayed on the browser's window.

#### The HEAD Section and the <title>, <meta>, <link>, <script> Elements

The HEAD section may contains these elements: <title>, <meta>, <link>, <script>, <base>.

The <title>...</title> container element encloses the title for the page, which shows up in bookmarks and history lists (the URL is stored if there is no title). Titles are used by search engines' to index your page.

The <meta> element contains meta-information, for use by browser to properly render the document. For example, <meta charset="utf-8"> specifies the character encoding scheme for the document.

You can use a <link> element to add an external CSS Style Sheet (and <style>...</style> element for internal styles):

<link href="filename.css" rel="stylesheet">

The <script>...</style> element is used to define programming scripts. For example, external JavaScript file:

<script src="filename.js"></script>

## HMTL visual elements

### Block vs Inline elements

Every HTML element has “display” property that specifies how the element is displayed. HTML elements can be classified by its default value (“inline”, “block”, “inline-block”) for this property, so there are 3 types of elements:

**Block elements:** A block element (such as <p>, <h1> to <h6> and <div>) starts on a new line, takes the full width, and ends with a new line. It is *rectangular* in shape with a *line-break* before and after the element and you can set the width, height.

**Inline elements (or Character Elements):** An inline element (such as <span>, <a>, <input>, <b>, <i>, <u>, <em>, <strong>, <code> and <span>) takes up as much space as it needs. It is *a continuous run of characters* and does not force a line-break before and after the element, although it can span a few lines. You cannot set the width, height of an inline element.

**Inline-Block elements**: An inline-block element (such as <img>, <video>, <iframe>) is an inline element (so it does not start on a new line) but you can set the width and height for it (i.e. like a block element).

Most There are only few inline elements (<strong>, <emp>, <code>, <q>) and the rest is block elements.

#### Block elements (most HTML elements are block)

**Heading <h?> (? Is level 1 to 6)**

**Paragraph <p>**

**Line break <br>**

**Horizontal rule <hr>**

**Pre-Formatted Text <pre>**

Function: Texts enclosed between <pre>...</pre> container tags are treated as pre-formatted, i.e., white space, tabs, new-line will be preserved and not ignored. The text is usually displayed in a fixed-width (or monospace) font. <pre>...</pre> is mainly used to display program codes. For example, my favorite Java's "Hello-world":

<pre>public class Hello {

public static void main(String[] args) {

System.out.println("Hello");

}

}</pre>

Becomes:

public class Hello {

public static void main(String[] args) {

System.out.println("Hello");

}

}

Without the <pre> tag, the entire program will be shown in one single line.

**Quote <blockquote>**

Function: Mark out a block of quote. Browsers typically indent the entire block to the right. For example,

<blockquote>Lorem ipsum dolor sit amet, consectetur adipisicing elit,

sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi

ut aliquip ex ea commodo consequat. Duis aute irure dolor in

reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Excepteur sint occaecat cupidatat non proident, sunt in culpa qui

officia deserunt mollit anim id est laborum.</blockquote>

#### Inline elements (a, img, video, iframe, input, button, strong, em, code, q, b, i, u, big, small, sub, sup)

The most commonly used inline elements are <a>, <input>, <img>, which is described later.

Logical-style formatting tags specify the semantic meaning (e.g., strong, emphasis, code); whereas physical-style formatting tags define the physical or typographical appearance (e.g., bold, italic, teletype). Logical styles should be used instead of physical styles. This is because physical styles deal with the appearance, which should be defined in style sheet, so as to separate the content and presentation.

**Logical-Style Formatting Tags**

The logical style character-level (inline) tags are:

|  |  |
| --- | --- |
| Logical-Style Tag | Meaning |
| <strong>...</strong> | strong emphasis (bold) |
| <em>...</em> | emphasis (italic) |
| <code>...</code> | program code (fixed-width monospace font) |
| <q>...</q> | quotation (enclosed in curly double quotes) |
| <ins>...</ins> | inserted |
| <del>...</del> | deleted |
| <def>...</def> | definition (bold or bold-italic) |
| <cite>...</cite> | citation (italic) |
| <kbd>...</kbd> | Keyboard (fixed-width monospace font) |
| <samp>...</samp> | sample text (fixed-width monospace font) |
| <abbr>...</abbr> | abbreviation (dotted underline, with title as tool tip) |
| <address>...</address> | address |
| <var>...</var> | variable (fixed-width or italic) |

The commonly-used tags are: <strong> (displayed in bold), <em> (displayed in italics), and <code> (use monospace font for programming codes).

Example:

<p>Lorem <q>curly quoted</q>, consectetur adipisicing elit,   
sed do <cite>citation</cite> incididunt ut labore et dolore magna aliqua.   
Ut enim ad minim veniam, quis <samp>sample</samp> exercitation ullamco laboris nisi   
ut <code>code</code> ex ea <kbd>keyboard</kbd> consequat. Duis aute irure dolor in   
reprehenderit in velit esse cillum dolore eu fugiat nulla pariatur.  
Excepteur <ins>insert</ins> occaecat <del>delete</del> non proident,

sunt in culpa qui officia deserunt mollit anim id est laborum.</p>

Lorem curly quoted, consectetur adipisicing elit, sed do citation incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis sample exercitation ullamco laboris nisi ut code ex ea keyboard consequat. Duis aute irure dolor in reprehenderit in velit esse cillum dolore eu fugiat nulla pariatur. Excepteur insert occaecat  non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

**Physical-Style Formatting Tags**

The following 7 physical-style formatting tags are still used in HTML5: <b>,<i>,<u>,<small>,<big>,<sup>,<sub>

|  |  |
| --- | --- |
| Physical-Style Tag | Meaning |
| <b>...</b> | Bold |
| <i>...</i> | Italic |
| <u>...</u> | Underline |
| <big>...</big> | large font |
| <small>...</small> | small font |
| <sup>...</sup> | Superscript |
| <sub>...</sub> | Subscript |

#### Inline-block elements (img)

<img> tag is an example of an inline-block element.

### List and Tables

#### List

##### Unordered List <ul> and List Item <li>

Function: An unordered list is shown with a bullet in front of each item. The <ul>...</ul> contains an unordered list. Each of items in the list is enclosed in <li>...</li>, as follow:

<ul>

<li>list-item-1</li>

<li>list-item-2</li>

......

</ul>

Example:

<p>An OMO web designer must master:</p>

<ul>

<li>Hypertext Markup Language (HTML)</li>

<li>Cascading Style Sheet (CSS)</li>

<li>HyperText Transfer Protocol (HTTP)</li>

<li>Apache HTTP Server</li>

</ul>

Output of the example:

An OMO web designer must master:

* Hypertext Markup Language (HTML)
* Cascading Style Sheet (CSS)
* HyperText Transfer Protocol (HTTP)
* Apache HTTP Server

You can use attribute type in <ul> tag to choose the style of the bullets:

* type="disc": a black dot (default).
* type="circle": an empty circle.
* type="square": a filled square.

##### Ordered List <ol> and List Item <li>

Items in an ordered list are numbered automatically by the browser. The container tag <ol>...</ol> contains an ordered list. Each item of the list is contained inside a <li>...</li> container tag. The syntax is similar to the unordered list.

You can use attribute start="number" in the <ol> tag to specify the starting number (which default to 1).

You can use the type attribute of the <ol> tag to choose the numbering style:

* type="1": numbers 1, 2, 3, ... (default)
* type="a": lowercase letters a, b, c, ...
* type="A": uppercase letters A, B, C, ...
* type="i": lowercase Roman numerals i, ii, iii, iv, ...
* type="I": uppercase Roman numerals I, II, III, IV, ...

#### Table

The basic unit of a table is a *cell*. Cells are grouped into *row*. Rows are grouped to form the *table*. This corresponds well to the "row-centric" approach in the display.

The essential tags used by tables are:

* <table>...</table>: contains the entire table.
* <tr>...</tr>: contains a row.
* <th>...</th> and <td>...</td>: contain a *header* cell and a *data* (*detail*) cell respectively.

Additional tags are:

* <caption>...</caption>: specifies a caption.
* <thead>...</thead>, <tbody>...</tbody>, <tfoot>...</tfoot>: marking out the table header, body and footer.
* <colgroup>...</colgroup> and <col>...</col>: for applying styles to column group and column respectively.

##### Entire table <table>, row <tr>, cell data <td>

**Entire Table <table>...</table>**

Function: Set up a table, consisting of rows.

Three optional presentation attributes, **border="n"** (specifies the width of borders, in pixels), **cellspacing="n"** (specifies the spacing between cells, in pixels), and **cellpadding="n"** (define the spacing between the content of the cell and its boundaries, in pixels), are often used in older HTML pages but now deprecated. The now-preferred approach is to use CSS (again! but coming soon!).

**Row <tr>...</tr>**

Function: Set up a row inside a table, consisting of cells.

**Data Cell <td>...</td> or Header Cell <th>...</th>**

Function: Set up each individual cell of a row (of a table). <th>...</th> defines a header cell (usually displayed in bold with center alignment) and <td>...</td> defines a body cell.

*An empty cell is typically marked as <td>&nbsp;</td>.*

For Example:

<table>

<caption>Price List</caption>

<tr>

<th>Fruit</th>

<th>Price</th>

</tr>

<tr>

<td>Apple</td>

<td>$0.50</td>

</tr>

<tr>

<td>Orange</td>

<td>$0.65</td>

</tr>

</table>

|  |  |
| --- | --- |
| Price List | |
| **Fruit** | **Price** |
| Apple | $0.50 |
| Orange | $0.65 |

##### Table Header <thead>, Table Body <tbody>  and Table Footer <tfoot>

Those tags act like <div>, <span>; they don’t make any visual effect but defining a header, body and footer sections for a table so that styles can be applied on them conveniently. Browser may duplicate the header or footer when breaking the table across multiple pages (in print-out).

**Column Group <colgroup>...</colgroup> and Column <col>**

An HTML table is row-centric. It consists of rows of cells. Nonetheless, you can also identify the columns via <colgroup> or <col>. But the features are really limited. Only a few properties could be applied on columns: border, background, width and visibility.

Function: <colgroup>...</colgroup> can be used to group a set of columns, so that styles can be applied to all the columns in the group. Similarly, <col> can be used to identify a column for applying styles.

The attribute span="numOfColumns" specifies the number of columns belonging to this <colgroup> or <col> declaration.

A <col> not nested under <colgroup> is automatically wrap under a <colgroup>.

Example:

<table>

<!-- colgroup 1 consists of col 1 and col 2 -->

<colgroup>

<col style="background-color:lightyellow" />

<col style="background-color:white" />

</colgroup>

<!-- colgroup 2 consists of col 3 and col 4 -->

<colgroup style="background-color:lightgrey" >

<col span="2" />

</colgroup>

<tr>

<td>Col 1 in the group 1</td>

<td>Col 2 in the group 1</td>

<td>Col 3 in the group 2</td>

<td>Col 4 in the group 2</td>

</tr>

<tr>

<td>Col 1 in the group 1</td>

<td>Col 2 in the group 1</td>

<td>Col 3 in the group 2</td>

<td>Col 4 in the group 2</td>

</tr>

</table>

|  |  |  |  |
| --- | --- | --- | --- |
| Col 1 in the group 1 | Col 2 in the group 1 | Col 3 in the group 2 | Col 4 in the group 2 |
| Col 1 in the group 1 | Col 2 in the group 1 | Col 3 in the group 2 | Col 4 in the group 2 |

##### "rowspan", "colspan" attributes to merge rows, columns

On a <td> or <th> cell, we can use the attribute rowspan="numOfRows" or colspan="numOfColumns" to span the cell to occupy multiple rows or columns. The subsequent <td> or <th> cells will adjust their positions accordingly.

Example 1:

<table>

<tr>

<td>11111</td>

<td>22222</td>

<td>33333</td>

</tr>

<tr>

<td>44444</td>

<td rowspan="2">55555</td>

<td>66666</td>

</tr>

<tr>

<td>77777</td>

<td>88888</td>

</tr>

</table>

|  |  |  |
| --- | --- | --- |
| 11111 | 22222 | 33333 |
| 44444 | 55555 | 66666 |
| 77777 | 88888 |

Example 2:

<table>

<tr>

<td colspan="2" rowspan="2">11111</td>

<td>22222</td>

</tr>

<tr>

<td>33333</td>

</tr>

<tr>

<td>44444</td>

<td>55555</td>

<td>66666</td>

</tr>

</table>

|  |  |  |
| --- | --- | --- |
| 11111 | | 22222 |
| 33333 |
| 44444 | 55555 | 66666 |

### Anchor: hyperlinks, bookmarks

<a> anchor element is used to set up a hyperlink to some webpage or a bookmark to some element in the document so that when users click on it, the browser will move to that link/bookmark.

**Setting up a Hyperlink <a href="url">...</a>**

<a href="http://www.w3c.org">W3C Home Page</a>

<a href="ftp://ftp.faqs.org">FTP to FAQS.ORG</a>

<a href="mailto:help@zzz.com">Email Help</a>

<a href="news:soc.culture.singapore">Singapore News</a>

Instead of displaying the targeted page pointed to by href in the *current* browser's window. You can use the attribute target="\_*blank*" to display the new page in another window.

**Setting up a Link Targeting a Specific HTML Element via the id Attribute**

<h1 id="ch1">Chapter 1</h1>

......

<h1 id="ch2">Chapter 2</h1>

......

Jump to <a href="#ch1">Chapter 1</a>

......

Jump to <a href="#ch2">Chapter 2</a>

## HTML grouping elements: div, span and Semantic Block Elements

### div and span

<div> block element and its counterpart, <span> inline element, are extensively used in the modern web pages to mark out a rectangular block or span of text that are used with CSS. Both div and span don’t have any visual effect.

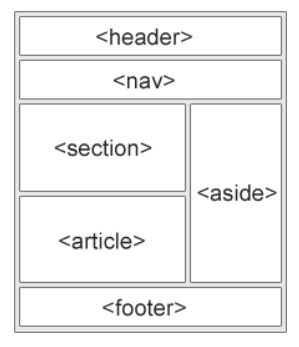
### HTML5 semantic block elements

Before HTML5, we rely on the all-purpose generic container <div> elements to structure a document into various sections and apply the formatting style:,

<div class="header"> ...... </div>

<div class="content"> ...... </div>

<div class="footer"> ...... </div>

This is less than desirable, as <div> elements do not provide semantic information about the sections.

HTML5 added many semantic block elements, which extends <div>, to structure a document. They are:

* <header>, <main>, <footer>,
* <article>, <section>,
* <aside>, <nav>
* <figure>, <figcaption>
* <summary>, <details>.

You are encouraged to replace some of the <div>'s with these more descriptive semantic elements.

#### <header>, <main>, <footer> and <article>, <section>

<header>......</header>

<main>

<article>

<section>......</section>

<section>......</section>

</article>

<article>

<section>......</section>

<section>......</section>

</article>

</main>

<footer>......</footer>

#### <figure> and <figcaption>

You can markup a figure via <figure>...</figure> that contains <figcaption>...</figcaption>, e.g.,

<figure>

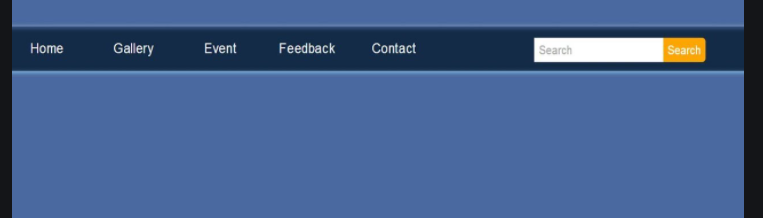
<img src="...." alt="...">

<figcaption>......</figcaption>

</figure>

#### Sidebar <aside> and navigation bar <nav>

The <aside> element can be used to introduce related contents, typically formatted in a floating sidebar alongside the main texts.



The <nav> element wraps a set of links into a menu. For example,

<nav>

<h1>....</h1>

<ul>

<li><a href="...">......</a></li>

<li><a href="...">......</a></li>

......

</ul>

</nav>

You can place the <nav> under an <aside> if the navigation menu is to be shown in a sidebar (or side panel).

## HTML forms

The input data from users is submitted to the server through HTML forms, indicated by <form>, which contains (many) input tags like <input>, <button>, <select>, <option>, <textarea>, <keygen> and label tag <label>.

### HTML forms, which contain input fields help submit user data to the server

#### <form> with action and method properties

An HTML form is enclosed by a <form> element.

The attribute **action** specifies the URL for which this web form is to be submitted, which *default to the current page*.

The attribute **method** specifies the HTTP method used for submission, i.e., GET or POST (*with default of GET*).

<form method="get|post" action="*url*">

... Labels for input elements ...

... Input elements ...

... Submit/Reset buttons ...

</form>

The input fields (or controls, widgets) and their labels are placed Inside the <form>.

#### Labeling a single input field:  <label>

Each <input> field is typically associated with a <label> to label the input field.

You can bind the <input> element it labels by:

* either matching the for attribute of the <label> element with the id attribute of the element
* or place the <input> element within the <label>...</label> element (without the for attribute)

If you want the input and label to go together in a same line, then put <input> element within the <label>

<form action="">

    <!-- Matching for with id-->

    <label for="username"> Username: </label>

    <input type="text" id="username" name="yourUserName">

    <br>

    <!-- Placing <input> inside <label> -->

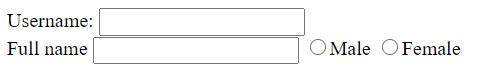
    <!-- Label can be in front or behind the <input> -->

    <label>Full name <input type="text" name="yourFullName"></label>

    <label><input type="radio" name="gender" value="m">Male</label>

    <label><input type="radio" name="gender" value="f">Female</label>

</form>



Label does not have any visual effect. But if you click on the label, the associated input field will be selected.

#### Labeling a group of input fields: <fieldset> and <legend>

A well-designed web form uses <fieldset> to group the input fields into sets, with a <legend> to provide a descriptive legend for the fieldset. For example

    <form>

        <fieldset>

            <legend>1st field set:</legend>

            Field one:<br>

            <input type="text"><br>

            Field two:<br>

            <input type="text"><br>

        </fieldset>

        <br>

        <fieldset>

            <legend>2nd field set:</legend>

            Field three:<br>

            <input type="text"><br>

            Field four:<br>

            <input type="text"><br>

        </fieldset><br>

        <input type="submit" value="Submit">

    </form>



### Input fields: <input>, <button>, <opt>, <textarea>, <keygen>

Input fields are marked by <input>, <textarea>, <select> and <button>.

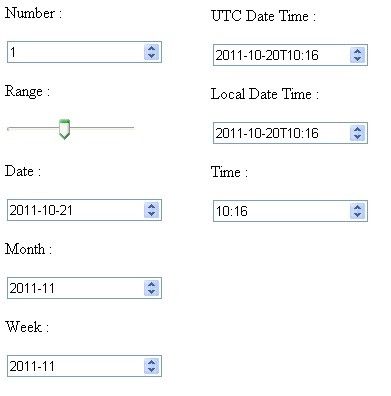
The most commonly used is <input> (a singleton element), which has attribute specifying sub-types (e.g., type="text" and type="checkbox").

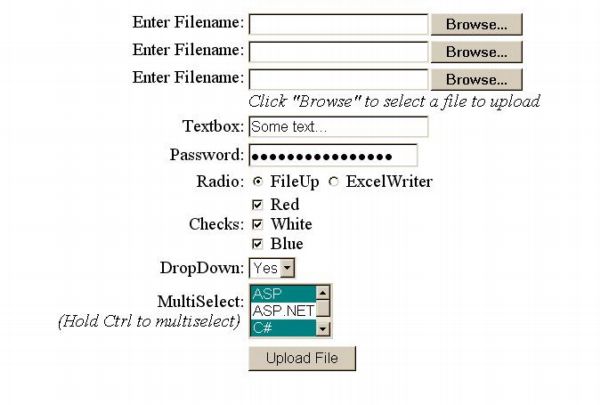


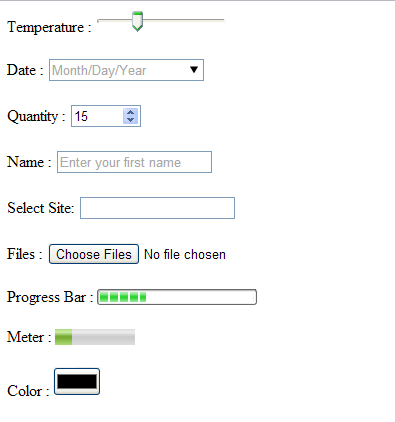
#### Radio and checkbox buttons

The <input> is a singleton element, with attribute specifying sub-types (e.g., type="text" and type="checkbox").

The <textarea>, <select>, <button> are container elements.







# CSS (Cascading Style Sheet)

HTML is the skeleton of a web page and CSS is the skin.

A *Style Sheet* is *a collection of style rules* that can be applied to *a selected set of HTML elements*. A style rule is used to control the appearance of HTML elements such as their font properties (e.g., type face, size and weight), color properties (e.g., background and foreground colors), alignment, margin, border, padding, and positioning. This is the same as the *styles* in any publishing software like WinWord or LaTex.

The word *cascading* means that multiple style rules can be applied to the same HTML element. The browser follows a certain *cascading order* in finalizing a style to format the HTML element in a predictable fashion.

## CSS syntax

### A CSS is a set of style rules for HTML documents

A CSS style sheet provides style rules to HTML documents. CSS is a language by itself, which is different from HTML, JS!

1. A style rule consists of a selector which selects the HTML elements it operates upon, and a list of style property name:value pairs enclosed in braces {...}, as follows:

selector {

property-name-1**:** property-value-1-1**,** property-value-1-2**,** ... **;**

property-name-2**:** property-value-2-1**,** property-value-2-2**,** ... **;**

......**;**

}

There are different kinds of selectors, for example:

body { /\* Apply to <body> and possibly its descendants \*/

font-family: "Segoe UI", Tahoma, Helvetica, Arial, Verdana, sans-serif;;

font-size: 14px;

margin: 10px auto 10px auto; /\* top right bottom left \*/

padding: 0;

}

This selector selects the <body> tag. Hence, the defined style is applied to the <body>...</body> element. Many (but not all) of the CSS properties (such as color, font) are inherited by its descendants, unless they are overridden by other style rules.

1. The name:value pairs are separated by semicolon ";". You can omit the last semi-colon before the closing brace "}". The name and value are separated by a colon ":" in the form of name:value.

Multiple values for the same property name are separated by commas "," (as in the font-family). However, multiple parts of the same property value are separated by space " " (as in the margin, which has a value with 4 parts).

1. Values containing space must be quoted, e.g., "Times New Roman" or 'Times New Roman'. Extra whitespaces (blank, tab and newline) are ignored.
2. If the same set of styles is applicable to more than one elements, the selectors can be grouped together in one single rule (called Group-Selector). The tagnames are separated by commas ",". For example, the following rule apply to elements <h1> to <h6>:

h1, h2, h3, h4, h5, h6 {

text-align: center;

font-family: "Trebuchet MS", "Segoe UI", Helvetica, Tahoma, Arial, Verdana, sans-serif;

}

1. Comments can be inserted inside the style sheet enclosed between /\* and \*/.

**Note: CSS Syntax is different from HTML Syntax**

CSS and HTML have different syntaxes!!! For example, HTML's attributes uses "=" to separate the name and value, in the form of name="value"; the name-value pairs are separated by spaces, as follows:

<img src="logo.gif" alt="Logo" height="10" width="20">

Also, the comment /\* \*/ of CSS is different from HTML’s <!-- … -->

### Types of Selectors

#### Basic selectors

##### Universal Selector: \*

The universal selector \* selects ALL the elements in the document. Example:

\* { margin:0; padding:0; } /\* all tags have margin and padding of 0 \*/

##### Element (Tag) Selector

The *tag-selector* selects all elements of the given tag-name. Example:

h2 { background-color:black; color:white; }

##### Class Selector: .C

The *Class Selector*, which begins with a dot '.' followed by the classname, selects all elements with the given classname, regardless of the tag name. Example:

.f14px\_i { font-size:14px; font-style:italic; }

.f16px\_b { font-size:16px; font-weight:bold; }

.red { color:red; }

.underline { text-decoration:underline; }

<p class="f14px\_i">Text is 14px and italic.</p>

<p class="f16px\_b">Text is 16px and bold.</p>

<p class="red">Text is in red.</p>

<h5 class="red">Text is in red.</p>

Take note that the class attribute may contain multiple values. This means that you can apply multiple class style rules to an HTML element. In the following example, second <p> belongs to 3 classes: f16px\_b, red, underline.

<p class="f14px\_i underline">Text is 14px and italic, and underlined.</p>

<p class="f16px\_b red underline">Text is 16px and bold, in red and underlined.</p>

##### ID Selector: #D

The ID-selector, begins with a '#' followed by the id value, selects a specific element with the given unique id value. Recall that the id value must be unique in an HTML document.

You can use <div>'s with unique id to divide the document into partitions of different styles. For example,

/\* ID selector for the 3 major division of the document \*/

#header { font-size:16px; align:center; font-style:bold; }

#header h1 { text-transform:uppercase; } /\* contextual selector \*/

#content { font-size:14px; align:justify; }

#content h3 { color:#FF0000; text-decoration:underline; } /\* red, underline \*/

#footer { font-size:12px; align:right; }

#footer p { color:#00FF00; text-decoration:none; } /\* green, not underline \*/

<body>

<div id="header">

<h1>H1 in the "header" division</h1>

<h3>H3 in the "header" division</h3>

<p>Paragraph in "header" division</p>

</div>

<div id="content">

<h1>H1 in the "content" division</h1>

<h3>H3 in the "content" division</h3>

<p>Paragraph in "content" division</p>

</div>

<div id="footer">

<p>Paragraph in "footer" division</p>

</div>

</body>

##### Tag-cum-ID T#D, Tag-cum-Class T.C selectors

The selector T.C selects all tag-name T with classname of C. This is a restricted form of the Generic-Class selector, which applies to the specific tag-name only.

An HTML element (such as <p>) can be sub-divided into different *style sub-classes* via the class attribute. This subclass mechanism allows us to apply different styles to different subclass of a particular element. For example,

p { color:black; } /\* default style for all <p> tags \*/

p.red { color:red; } /\* applicable to <p class="red"> tags (override default) \*/

p.blue { color:blue; } /\* applicable to <p class="blue"> tags (override default) \*/

<p>This paragraph is in black (default style)</p>

<p class="red">This paragraph, of class="red", is in red.</p>

<p class="blue">This paragraph, of class="blue", is in blue.</p>

For the selector T#D, the part T seems redundant since #D identifies uniquely an element; it’s there to serve documentation purpose.

##### Attribute Selectors

* T[att]: selects elements that possess the given attribute, regardless of value.
* T[att="value"]: selects elements that possess the given attribute, with the given value.

##### Group Selector: S1, S2

You can apply the same style definitions to multiple selectors, by separating the selectors with a commas ','.

selector-1, selector-2, ... { style-definitions }

The selector-x could be any kind of selectors, such as Tag-selector, Class-selector, or ID-selector. Example:

h1, .normal, #footer { background-color:black; color:white; }

#### Descendant, Child, Sibling selectors: space, >, :, +, ~

You can define a style rule that takes effect only when a tag occurs within a certain contextual structure, e.g., descendant, immediate-child, first-child, sibling, etc.

##### Descendant T1 T2

To create a descendant selector, list the tags in their hierarchical order, with no commas separating them (commas are meant for grouping selectors). Descendant means child, grandchild, i.e. any level descendant. Example:

div p { color:red; }

applies on

<div><table><tr><td><p> <!...

##### Any Child Selector T1 > T2 and First Child T1 : T2

T1 > T2 {color:red;}/\* The style is applied to T2 element if T2 is an (immediate) child of T1\*/

T1 : T2 {color:red;}/\*The style is applied to T2 element if T2 is the first (immediate) child of T1\*/

##### Any Sibling (Adjacent) Selector T1 + T2, Immediately Following Sibling T1 ~ T2

/\*Style is applied to T2 if T1, T2 share a same immediate parent and have same position\*/

T1 + T2 {color:red;}

/\*Style is applied to T2 if T2 follows immediately after T1, and T1,T2 share a same immediate parent\*/

T1 ~ T2 {color:red;}

#### Pseudo selectors

##### Pseudo-class selector a:link|a:visited and :focus|:hover|:active

CSS defines a number of pseudo-classes for anchor elements <a>, namely, a:link (unvisited link), a:visited (visited link), a:focus (on focus), a:hover (mouse pointer hovers over), a:active (clicked or active link). Take note that colon ":" is used to identify pseudo classes instead of "." for ordinary classname.

These pseudo classes is commonly-used with the <a> element. But :hover, :focus, and :active can also be applied to other elements, such as <p>, <li>, and etc.

Example:

a { font-size:14px; } /\* all <a> tags \*/

a:link { color:red; } /\* unvisited link \*/

a:visited { color:green; } /\* visited link \*/

a:focus { color:lightblue; } /\* on focus via tab key \*/

a:hover { color:blue; } /\* mouse over link \*/

a:active { color:black; } /\* currently selected link \*/

##### Pseudo-Elements Selector :before and :after (::before and ::after in CSS3)

The :before and :after pseudo-elements lets you select elements and add content before or after the elements.

.title:before { content: "\0022"; }

.title:after { content: "\0022"; }

The above rules will add a double-quote (Unicode 0022H) in front and behind all elements having class="title".

Take note that these selectors generate contents!

### 3 places for CSS: Inline (style = “”), Internal (<style>…<style>), External (<link rel=”stylesheet” href=””>)

There are three places where you can define style rules: inline, internal and external.

Inline styles have the highest priority, followed by internal styles, and then external styles.

**Inline Style**: Included inside a particular HTML opening tag's via attribute **style="*style-rules*"**. The rules are applicable to that particular HTML element only.

<!DOCTYPE html>

<html>

<body>

<p style="font-size:18px; font-family:cursive">This paragraph uses 18px cursive font.</p>

<p>This paragraph uses default font.</p>

<p>This paragraph uses <span style="font-size:20px">20px inside this span</span>

but default font size here.</p>

</body>

</html>

**Internal (Embedded) Style Sheet**: Embedded inside the **<style>...</style>** tags in the HEAD section of the HTML document. The styles are applicable to that entire document.

<!DOCTYPE html>

<html>

<head>

<style>

body { background-color:cyan }

h2 { color:white; background-color:black }

p.cursive { font-size:18px; font-family:cursive }

p.f20px { font-size:20px }

</style>

</head>

<body>

<h2>H2 is white on black</h2>

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

<p class="cursive">This paragraph uses 18-px cursive font.</p>

<p class="f20px">This paragraph uses 20-px font.</p>

</body>

</html>

**External Style Sheet (Recommended)**: Stored in an external file, which is then linked to HTML documents via a **<link href="External.css" rel="stylesheet">** element (“rel” means relationship) in the HEAD section. The main advantage of external style sheets is that the same set of styles can be applied to all HTML pages in your website to ensure uniformity in presentation.

/\* testExternal.css \*/

body { background-color:cyan; color:red; }

h2 { background-color:black; color:white; text-align:center; }

p { font-size:12pt; font-variant:small-caps; }

p.f24pt { font-style:italic; font-size:24pt; text-indent:1cm; }

#green { color:green; }

This HTML document references the external style sheet via the <link> element in the HEAD section:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<link href="TestExternal.css" rel="stylesheet">

</head>

<body>

<h2>H2 is white on black</h2>

<h2 id="green">This H2 is green on black</h2>

<p>The default paragraph uses 12-pt small-cap font.</p>

<p class="f24pt">This paragraph uses 24-pt, italics font with text-indent of 1cm.

It inherits the small-cap property from the default paragraph selector.</p>

</body>

</html>

### Inheritance

Many (but not all) CSS properties, such as color and font-family, affect not only the elements selected by the selector, but also inherited by their descendants.

Inheritance is a big time-saver for designing styles. For example, you set the default color and font-family in the <body> element, which will then be inherited by all the elements. You then override the default properties for specific elements, if needed.

Some properties (such as border, margin, padding, width, height, background-color) are not inherited. This is for good reason. For example, if border is defined for <ul> and is inherited, then all descendants (<li>) will be drawn with border!

**Special Property Value: inherit**

You can use a special property-value called "inherit" to inherit the property from its ancestor. For example,

<!DOCTYPE html>

<html>

<head>

<style>

p { border: 5px solid red; }

.inherit-border { border: inherit; }

</style>

</head>

<body>

<p>The <em>border</em> property is not inherited.</p>

<p>The <em class="inherit-border">border</em> property is inherited.</p>

</body>

</html>

Although the first <em> is nested under the <p> tag, the border property is not inherited from the ancestor. That is, you will not see a border around the <em>'s content. We can force the inheritance by assigning a special value "inherit" as shown in the Class Selector .inherit-border.

### Style Conflict

Style conflict on an element arises when:

1. A property is inherited from multiple ancestors.
2. More than one rules are applicable to the element. For example, Tag-selector p, Class-selector .red and ID-selector #comment are all applicable to element <p id="comment" class="red">.

#### Nearest Ancestor Wins

If a property is not defined for an element and is inheritable, it will be inherited from the nearest ancestor.

#### More specific selector wins

The more specific the selector, the stronger the rule. For example,

<!DOCTYPE html>

<html>

<head>

<style>

p { color:black; background-color:white; }

/\* Override the color properties \*/

p.red { color:red; }

p#id1 { color:yellow; background-color:lightblue; }

p#id2 { color:blue; }

p#id1 { color:green; }

</style>

</head>

<body>

<p id="id1">Paragraph with id of "id1" (green)</p>

<p id="id2">Paragraph with id of "id2" (blue)</p>

<p class="red">Paragraph of class of "red" (red)</p>

<p id="id1" class="red">Paragraph with id of "id1" and class of "red" (green)</p>

<p>Paragraph without id and class with default colors (black)</p>

</body>

</html>

The p Tag-selector is the most general, which selects all the <p> elements; the p.red Class-selector selects a class of <p> elements with attribute class="red"; the p#id1 and p#id2 ID-selectors select only one element each with a unique id value. The ID-selector is the most specific (and takes precedence); followed by Class-selector; and followed by the general Tag-selector.

#### Last style wins and priority: inline > internal > external

The style-rule that read in last by the browser takes effect. In the above example, there are two ID-selector for id1, the latter takes effect.

The inline style (style attribute) overrides the internal style (defined in <style>) and external style sheet (defined via <link>). For internal and external styles, the order of <link> and <style> elements determine the precedence. It is recommended to place the <link> before <style> so that the internal styles can override the external styles.

#### “!important” wins all

You can override all the cascading rules by appending a special property-value !important, e.g.,

p { color:blue !important; background-color:grey; } /\* color cannot be overridden \*/

p { color:red; background-color:lightblue; } /\* override background-color only \*/

<p>color is blue but background is lightblue</p>

## CSS units: color and length

#### Color Properties

##### Color types: RGB, RGBA, HSL, HSLA

Color can be expressed as:

1. RGB hexadecimal triplets in the form of #rrggbb, where rr, *gg*, *bb* are values of red, green and blue. The values are between 00 and FF, in hexadecimal. For example, #12ABFF. The color value #112233 can be shorthand as #123.
2. RGB in the form of rgb(r, g, b). The *r*, *g*, *b* can be expressed in a decimal value between 0 and 255; or in percentage between 0% and 100%.
3. RGBA in the form of rgba(r, g, b, a): RGB with an additional A (alpha channel). The A is used to control the transparency/opacity, with a=1 for opaque; and a=0 for totally transparent.
4. HSL in the form hsl(hue, saturation, lightness): Hue is the color on the color wheel in degrees between 0 to 360. Saturation (purity of color) is expressed in percentage between 0% and 100% (pure color). Lightness (brightness or intensity) is also expressed in percentage between 0% (darkest) and 100% (brightest).
5. HSLA in the form of hsla(hue, saturation, lightness, alpha).
6. The 16 pre-defined English color names as follows. These 16 colors are numerically generated and are really really ugly. You should avoid using them!! Many browsers also support other color names such as lightblue, lightgreen, etc.

##### CSS's Color Properties

The most important color properties are color and background-color:

* color:#rrggbb|rgb(r,g,b)|rgba(r,g,b,a)|color-name  
  Set the color of the text (or foreground). Color values are inherited by descendants.
* background-color:#rrggbb|rgb(rrr,ggg,bbb)|rgba(r,g,b,a)|color-name|transparent  
  Set the background color of an element. The default is transparent and NOT inherited, so as to create a see-through effect.

#### Length Measurements

Many CSS properties, such as width, height, margin, border, padding, font-size and line-height, require a length measurement. For example,

p {

width: 80%; /\* 80% of the parent's width \*/

margin: 10px; /\* pixels \*/

border: 5mm; /\* millimeters \*/

padding: 0;

font-size: 1.2em; /\* 1.2 times of the parent's font-size \*/

line-height: 1.5; /\* 1.5 times of the current font-size \*/

}

There shall be no space between the number and the unit, as space is used to separate multiple values.

There are two types of length measurements: relative (to another length property) and absolute (e.g., inches, centimeters, millimeters).

##### Absolute units

* **in** (inch)
* **cm** (centimeter)
* **mm** (millimeter)
* **pt** (point): 1 inch has 72 points. 1pt is 1/72 in ≈ 0.014in ≈ 0.35mm.
* **pc** (pica): 1 pica is 12 points. 1 inch has 6 picas. 1pc ≈ 1/6 in ≈ 0.17in ≈ 4.2mm. pc is not commonly used.

##### Relative units

rem Relative to the font-size of the root element

em Relative to the font-size of the current element for properties like width. In the case of font-size, em is relative to the font-size of the parent element.

vw Relative to 1% of the width of the viewport\*

vh Relative to 1% of the height of the viewport\*

vmin Relative to 1% of viewport's\* smaller dimension

vmax Relative to 1% of viewport's\* larger dimension

% Relative to the **parent** element

p {

width: 80%; /\* 80% of the parent's width \*/

font-size: 1.2em; /\* 1.2 times of the parent's font \*/

margin: 1.2em; /\* 1.2 times of the current font's letter 'm' \*/

padding: 10px; /\* 10 pixels \*/

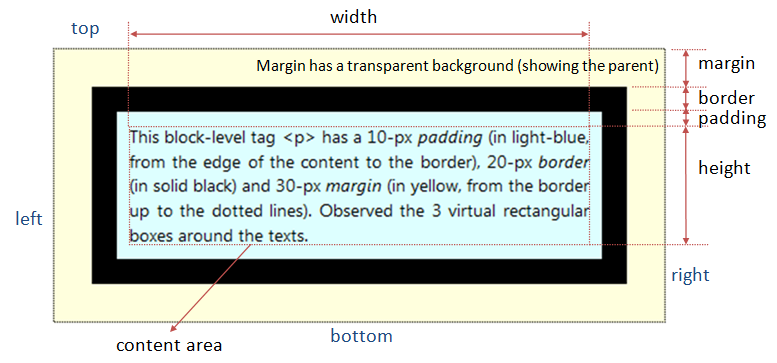
border: 0; /\* zero does not need a unit \*/

}

## Layout with CSS

### Box (margin, border, padding, width/height): every block element is a box

A block element (such as <p>, <div>, <h1>) is always *rectangular* in shape and exhibits the so-called *box model*, with *four CONCENTRIC virtual rectangles* wrap around its "*content area*", representing the *content area*, *padding*, *border*, *margin*, as illustrated below.



#### width, height: auto (fit child), 100% (fit parent)

The dimensions for the content area are specified in units (nu) such as px (pixels), or *percent* relative to the parent element (n%).

* width: auto|*nu*|*n*%  
  height: auto|*nu*|*n*%
* max-width: none|*nu*|*n*%  
  max-height: none|*nu*|*n*%  
  min-width: none|*nu*|*n*%  
  min-height: none|*nu*|*n*%

The max-width property defines the maximum width of an element. If the content is larger than the maximum width, it will automatically change the height of the element. If the content is smaller than the maximum width, the max-width property has no effect. The value of the max-width property overrides the width property so the width cannot exceed max-width no matter what you set width.

Note: *auto (fit child), 100% (fit parent)*

##### width and height are for content area excluding padding, border, margin

The **width** and **height** specify its content area, exclude the margin, border and padding. To get the *actual* size of the element, you need to add the margin, border and padding to the width/height. Example

#elm {

width: 300px;

margin: 10px;

border: 5px solid black;

padding: 20px;

}

/\* The *actual width* of the element is 300+(10+5+20)x2 = 370px \*/

##### width:auto

For most of the block elements (e.g., <div>, <p>), the default of width:auto sets the width to the width of the parent minus its own (margin+border+padding). Images <img> have an auto width equals to its actual width. Float elements have auto width of 0.

#### padding, margin (margin is used to create space space around elements – need to note it here)

Padding and margin are specified by 4 parameters: top, right, bottom, left. The values should be in units (nu) such as px (pixels), or *percent* relative to the parent element (n%).

* margin-top: auto|*nu*|*n*%  
  margin-right: auto|*nu*|*n*%  
  margin-bottom:auto|*nu*|*n*%  
  margin-left: auto|*nu*|*n*%  
  or shorthand form:

margin: *margin-top margin-right margin-bottom margin-lef*t  
margin: *margin-top-bottom margin-right-left*  
margin: *all-4-margins*

* padding-top: *nu*|*n*%  
  padding-right: *nu*|*n*%  
  padding-bottom: *nu*|*n*%  
  padding-left: *nu*|*n*%  
  or shorthand form:

padding: *padding-top padding-right padding-bottom padding-left*  
padding: *padding-top-bottom padding-left-right*  
padding: *all-4-padding*

##### Margin-right automatically fits the parent’s width

Browser would automatically adjust the margin-right to fill the container's width if the sum of its width, left and right margin/border/padding does not add up to the full width of the containing element. Take note that browser will not adjust the width, padding-right, border-right and the left margin/border/padding.

##### Center a block element by auto margin-left, margin-right

To center a block element, you set the margin-left and margin-right to auto (browser divides the remaining width to left and right margins equally).

For example, all the selected <div> are centered:

div#header {

margin: 10px auto; /\* 20px for top and bottom, auto for left and right \*/

}

div#footer {

margin: 10px auto 5px auto /\* top right, bottom, left \*/

}

div#content {

margin-top: 10px;

margin-right: auto;

margin-bottom: 5px;

margin-left: auto;

}

#### border

* border-width: thin|medium|thick|*nu /\*nu is number + unit\*/*
* border-style: none|hidden|dotted|dashed|solid|double|groove|ridge|inset|outset  
  border-color: #*rrggbb*|rgb(*r*,*g*,*b*)|rgba(*r*,*g*,*b*,*a*)|*color-name*

The 3 properties above can be combined into one property

* border: *border-width border-style border-color*

Example: every element (e.g. table) doesn’t have a visible border, to make a border visible, use, for example:

border: 1px solid black

### Normal flow and tools to alter overview

Normal flow is how the browser lays out HTML pages by default when you do nothing to control page layout. Let's look at a quick HTML example:

|  |  |
| --- | --- |
| <p>I love my cat.</p>  <ul>  <li>Buy cat food</li>  <li>Exercise</li>  <li>Cheer up friend</li>  </ul>  <p>The end!</p> | I love my cat.   * Buy cat food * Exercise * Cheer up friend   The end! |

Note here how the HTML is displayed in the exact order in which it appears in the source code, with elements stacked up on top of one another — the first paragraph, followed by the unordered list, followed by the second paragraph.

(recall: The elements that appear one below the other are described as block elements, in contrast to inline elements, which appear one beside the other, like the individual words in a paragraph.)

For many of the elements on your page the normal flow will create exactly the layout you need, however for more complex layouts you will need to alter this default behavior using some tools:

* The [display](https://developer.mozilla.org/en-US/docs/Web/CSS/display) property
* Flexbox
* Grid
* Floats
* Positioning
* Table layout
* Multiple-column layout

### The old way: position and float

#### Positioning with top, left, bottom, right in absolute/relative/fixed manner

Each element has a natural location inside a page's flow, in the order read in by the browser. The property position can be used to alter the position of block elements.

position: static|absolute|relative|fixed

By default, elements are displayed from top to bottom in the normal flow. For block elements, line breaks are inserted at the beginning and the end to form a rectangular box. You can leave the box in the default normal flow (position:static); you can remove the box from the normal flow and specify its location with respect to either its parent element (position:absolute) or the browser window (position:fixed) or its normal position in the flow (position:relative).

For non-static positioned elements, the new position is specified via top, left, bottom, right, width, height properties:

* top: n|n%|auto  
  left: n|n%|auto  
  bottom: n|n%|auto  
  right: n|n%|auto  
  Set the distance from the edge of this element to the corresponding edge of the containing block.
* width: n|n%|auto  
  height: n|n%|auto  
  Set the width and height of this block. You can use the width and height to scale this block.
* z-index: number|auto  
  When two blocks overlap due to re-positioning, the one with larger z-index number is on top (i.e., the z-axis in 3D graphics coordinates system). Negative number is allowed. The default auto stacks the element at the same level as its parent. If the z-index of two elements are the same or no z-index are defined, the last element rendered is placed on top. z-index with alpha can create see-thru effect.

**Note: the top, left, bottom, right values here are different from the top, right, bottom, left values in padding, margin!**

##### position:static and top, right, bottom, left have no effect

The default position:static positions the element according to the normal flow of the page, in the order that is read by the browser. Properties top, right, bottom, left has no effect for static.

<https://developer.mozilla.org/en-US/docs/Web/CSS/position> --> example of use

##### position:relative

Move the element relative to its normal location. The original space occupied by this element is preserved. The surrounding elements are not affected. For example,

div#up {

position: relative;

top: -2em; /\* move this element up by 2em \*/

}

##### position:absolute for child and position:relative for parent

Position the element relative to the first non-static ancestor element; or <body> if no such element is found. Absolute-positioned element is taken out from the normal flow, as if it does not present.

To absolutely position an element in a containing element (other than <body>), declare the containing element relative without any movement, e.g., container { position:relative }.

For example,

#left-panel {

position: absolute;

left: 10px; /\* from left edge \*/

top: 10px; /\* from top edge \*/

width: 200px;

height: 600px;

background: black;

color: white;

}

#main-panel {

position: absolute;

left: 220px; /\* from left edge 10+200+10 \*/

top: 10px; /\* from top edge, same as left-panel \*/

width: 560px;

height: 600px;

background: cyan;

color: black;

}

<body>

<div id="left-panel">

<h2>Left Panel</h2>

<p>This paragraph is on the left panel</p>

</div>

<div id="main-panel">

<h2>Main Panel</h2>

<p>This paragraph is on the main panel</p>

</div>

</body>

Absolute positioning repeatedly can be used to create animation.

##### position:fixed and it does not scroll away

The element is fixed at the position relative to the browser's window, and it does not scroll away. The position is defined in top, left, bottom, right (or width and height) properties. Fixed-positioned element is taken out of the normal flow, as if it is not present.

For example, a fixed <div> is added to the above example in absolute positioning. Take note that z-index is used to ensure that the fixed <div> is always on top of the other <div>'s, regardless of the order of writing the <div>'s.

#left-panel {

position: absolute;

left: 10px; /\* from left edge \*/

top: 10px; /\* from top edge \*/

width: 200px;

height: 600px;

background: black;

color: white;

z-index: 100;

}

#main-panel {

position: absolute;

left: 220px; /\* from left edge 10+200+10 \*/

top: 10px; /\* from top edge, same as left-panel \*/

width: 560px;

height: 600px;

background: cyan;

color: black;

z-index: 100;

}

#fixed-panel {

position: fixed;

left: 100px;

top: 200px;

width: 200px;

height: 200px;

background: lime;

color: red;

z-index: 200; /\* larger z-index is on top \*/

}

<body>

<div id="left-panel">

<h2>Left Panel</h2>

<p>This paragraph is on the left panel</p>

</div>

<div id="main-panel">

<h2>Main Panel</h2>

<p>This paragraph is on the main panel</p>

</div>

<div id="fixed-panel">

<h2>Fixed Panel</h2>

<p>This panel does not move when your scroll up/down.</p>

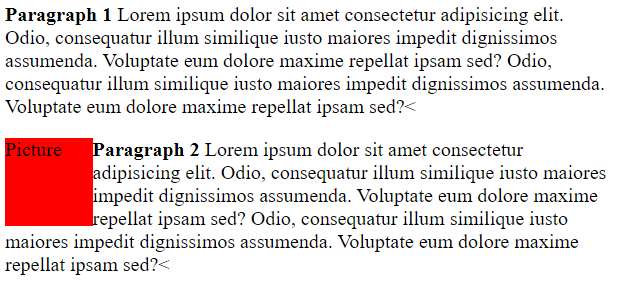
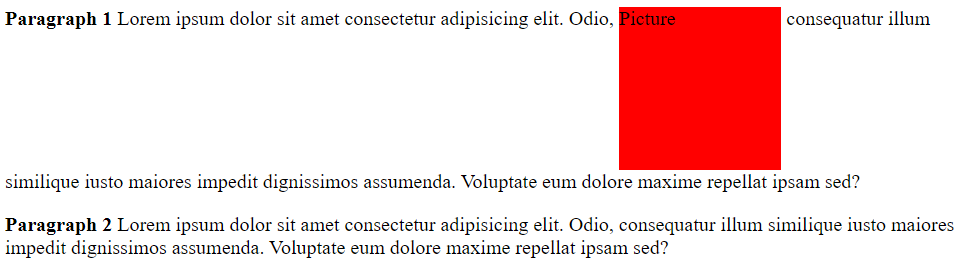
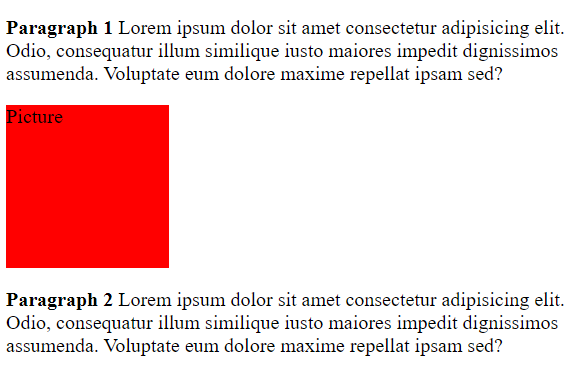
</div>

</body>

#### Float

If you want to insert a picture between two paragraph so that the text surround the picture, it’s likely that you will end up in: either (1) the picture is a block element separating two paragraphs or (2) both paragraph 1 and the picture are an inline elements and they make a gap between lines in paragraph 1.

“float” property solve this situation by letting the picture to occupy the space it needs and push paragraph 2 to the right.

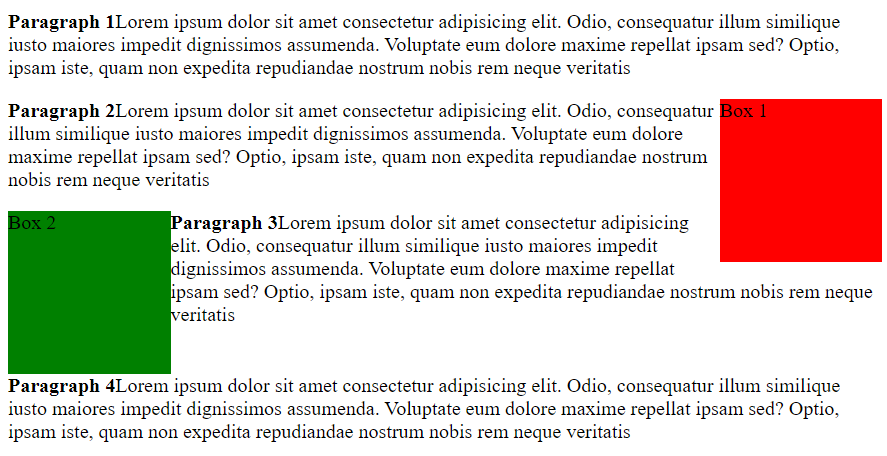


float: left|right|none

If an element is set “float: left”, then the element below it, is pushed to the right so that the “float” element can stand on the left of the element below. Note that the element above the “float” element is not affected; only the element below is.

An element can be float only horizontally (i.e., left and right), not up and down. Other than images, a float element shall have the width and height explicitly specified. Float elements are actually taken out of the normal flow. The following element acts as if the floated element is not there, but the enclosing texts would wrap around the floated element.

clear: left|right|both|none

If an element is set “clear: left”, it doesn’t allow the element above it to float on the left. As a result, the “clear:left” element always stays on the left margin of its parent

Example: a document that has the following flow will be presented as the figure

* paragraph 1,
* box 1 with “float: right”,
* paragraph 2,
* box 2 with “float: left”,
* paragraph 3,
* paragraph 4 with “clear: left”.

### Flexbox

The Flexible Box Module, usually referred to as flexbox, was designed as a one-dimensional layout model, and as a method that could offer space distribution between items in an interface and powerful alignment capabilities. (One dimensional layout means laying out elements either as a row or as a column, which is contrasted with the two-dimensional model of [CSS Grid Layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout), which controls columns and rows together)

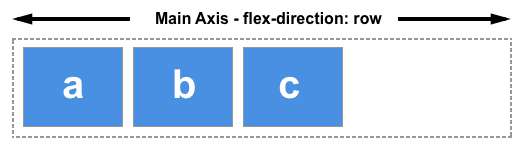
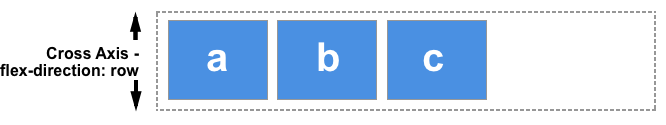
#### main axis, cross axis, start, end

flexbox features properties that align and justify content along two axes — the main axis and the cross axis. The main axis is defined by the [flex-direction](https://developer.mozilla.org/en-US/docs/Web/CSS/flex-direction) property, and the cross axis runs perpendicular to it.

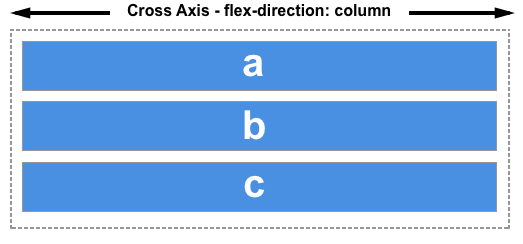
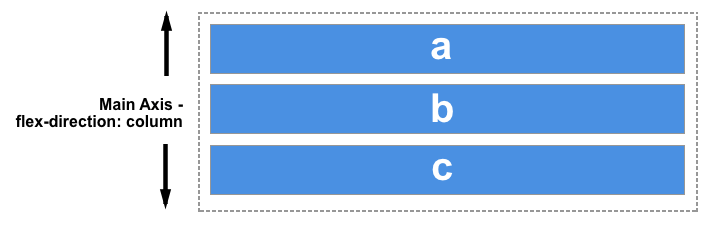
The main axis is defined by flex-direction, which has four possible values:

* row
* row-reverse
* column
* column-reverse

Should you choose row or row-reverse, your main axis will run along the row in the **inline direction**, and the cross axis runs down the columns.

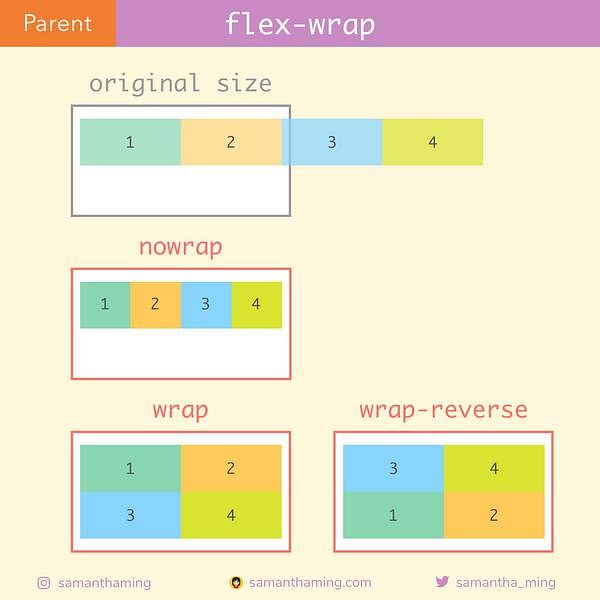
 

Choose column or column-reverse and your main axis will run from the top of the page to the bottom — in the **block direction** and the cross axis runs along the rows.



The start and end of the main axis is decided by the language the html uses, so for English and flex-direction=row, the start is the left, the end is the right. But for Arabic, flex-direction=row, the start is the right, the end is left.

#### The flex container

An area of a document laid out using flexbox is called a **flex container**. To create a flex container, we set the value of the area's container's [display](https://developer.mozilla.org/en-US/docs/Web/CSS/display) property to flex (block element) or inline-flex (inline-element). As soon as we do this the **immediate children** of that container become **flex items**. As with all properties in CSS, some initial values are defined, so when creating a flex container all of the contained flex items will behave in the following way.

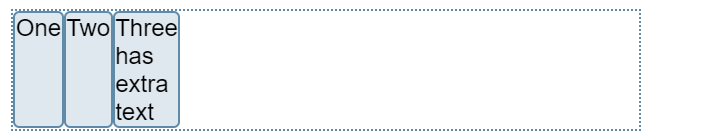
* Items display in a row (the flex-direction property's default is row).
* The items start from the start edge of the main axis.
* The items do not stretch on the main axis, but can shrink.
* The items will stretch to fill the size of the cross axis.
* The [flex-basis](https://developer.mozilla.org/en-US/docs/Web/CSS/flex-basis) property is set to auto.
* The [flex-wrap](https://developer.mozilla.org/en-US/docs/Web/CSS/flex-wrap) property is set to nowrap.

The result of this is that your items will all line up in a *row*, using the size of the content as their size in the main axis. If there are more items than can fit in the container, they will not wrap but will instead **overflow**. If some items are taller than others, all items will stretch along the cross axis to fill its full size.

.box {

display: flex;

}

<div class="box">

<div>One</div>

<div>Two</div>

<div>Three

<br>has

<br>extra

<br>text

</div>

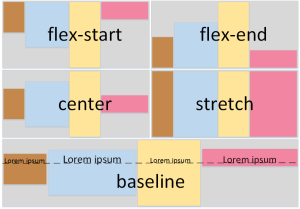
</div>

The flex-wrap: wrap|nowrap will force all the flex items into one row accepting overflow if the items cannot shrink enough (nowrap) or let the items to flow into multiple rows (wrap) – see the figure above.

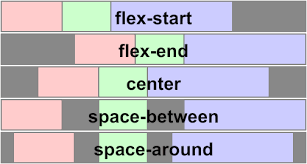
##### Alignment, justification and distribution of free space between items

A key feature of flexbox is the ability to align and justify items on the main- and cross-axes, and to distribute space between flex items. Note that these properties are to be set on the **flex container, not on the items** themselves.

**align-items for cross axis:** The [align-items](https://developer.mozilla.org/en-US/docs/Web/CSS/align-items) property will align the items on the **cross axis**. The default value for this property is stretch and this is why **flex items stretch to the height of the flex container by default**. This might be dictated by the height of the tallest item in the container, or by a size set on the flex container itself.



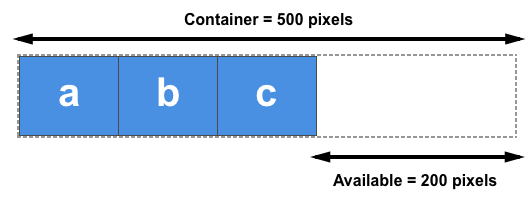
**justify-content for main axis:** The [justify-content](https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content) property is used to align the items on the **main axis**, the direction in which flex-direction has set the flow. The initial value is flex-start which will line the items up at the start edge of the container.



#### The flex items

The 3 properties flex-grow, flex-shrink, flex-basis defines how to distribute the **leftover space** between the items.

What is available space? If we have three 100 pixel-wide items in a container which is 500 pixels wide, then the space we need to lay out our items is 300 pixels, which leaves 200 pixels of available space. If we don’t change the initial values then flexbox will put that space after the last item.



##### The flex-basis property

The flex-basis defines the size of an item. **The initial value of this property is auto — in this case the browser looks to see if the items have a size.** In the example above, all of the items have a width of 100 pixels and so this is used as the flex-basis.

**If the items don’t have a size then the content's size is used as the flex-basis.** This is why when we just declare display: flex on the parent to create flex items, the items all move into a row and take only as much space as they need to display their contents.

##### The flex-grow property

With the flex-grow property set to a positive integer, flex items can grow along the main axis from their flex-basis. This will cause the item to **stretch** and take up any available space on that axis, or a proportion of the available space if other items are allowed to grow too.

If we gave all of our items in the example above a flex-grow value of 1 then the available space in the flex container would be equally shared between our items and they would stretch to fill the container on the main axis.

The flex-grow property can be used to distribute space in proportion. If we give our first item a flex-grow value of 2, and the other items a value of 1 each, 2 parts will be given to the first item (100px out of 200px in the case of the example above), 1 part each the other two (50px each out of the 200px total).

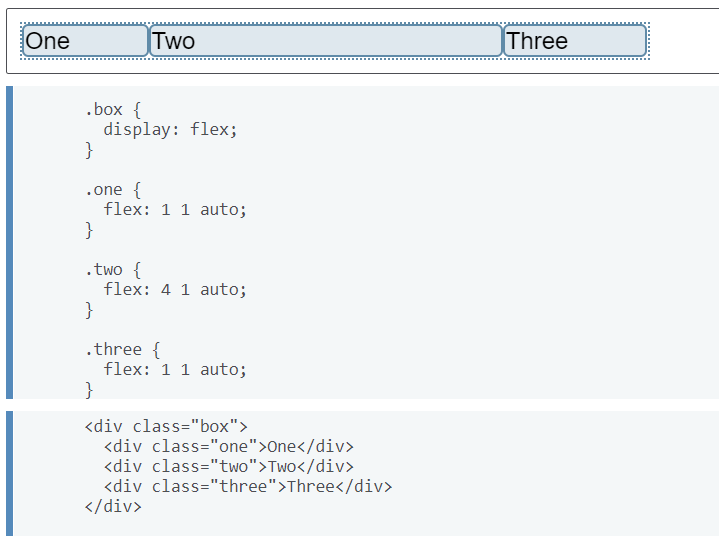
##### The flex-shrink property

If we do not have enough space in the container to lay out our items, and flex-shrink is set to a positive integer, then the item can become smaller than the flex-basis. As with flex-grow, different values can be assigned in order to cause one item to shrink faster than others — an item with a higher value set for flex-shrink will shrink faster than its siblings that have lower values.

##### Shorthand values for the flex properties

The flex-grow, flex-shrink, and flex-basis properties are often combined into the [flex](https://developer.mozilla.org/en-US/docs/Web/CSS/flex) shorthand, like:

flex: 1 2 auto



There are also some predefined shorthand values which cover most of the use cases:

* flex: initial /\* flex: 0 1 auto \*/

flex-grow is 0, so items will not grow larger than their flex-basis size.

flex-shrink is 1, so items can shrink if they need to rather than overflowing

flex-basis is auto. Items will either use any size set on the item in the main dimension, or they will get their size from the content size.

* flex: auto /\* flex: 1 1 auto \*/

everything is as with flex:initial but in this case the items can grow and fill the container as well as shrink if required

* flex: none /\* flex: 0 0 auto \*/

The items cannot grow or shrink but will be laid out using flexbox with a flex-basis of auto.

* flex: <positive-number>

Exp: flex: 1 or flex: 2 and so on. This is as if you used flex: 1 1 0. The items can grow and shrink from a flex-basis of 0.

#### negative margin

#### z-index

### Layout with float, position, flexbox

#### position property: relative, fixed, absolute

#### float and clear float

#### flexbox

### Layout with css grid

### Media queries and responsive

## Animation and Transition with CSS

<https://www.taniarascia.com/overview-of-css-concepts/>

This guide covers all the fundamentals of CSS - from syntax, selectors, and specificity to layouts and responsive media queries.

<https://zellwk.com/blog/9-important-css-properties-you-must-know/>

https://www.quora.com/What-are-the-important-concepts-in-CSS

The most important concepts which i see are as follows and they would help you in understanding the layout. The colors, fonts, background etc. can be learned once these concepts are understood properly.

* Position(fixed, relative etc.)
* Float concept(left, right, no float, inheritance in float)
* Concept of margin and padding
* Negative margins
* Z-index

I would start with the

1. Selectors
2. css box model
3. Media Queries and breakpoints
4. Flexbox
5. CSS grids or tables
6. Block, inline elements
7. Floats and clear floats
8. Responsive images
9. Web fonts

Categories -

* Text styling
* Element styling
* Media

**The important principles of CSS are as follows:**

* **CSS** Float Property.
* Negative Margins.
* Equal Height Column Layouts With Borders And Negative Margins In **CSS**.
* Z-Index **CSS** Property.
* **CSS** Specificity And Inheritance.
* ! **important CSS** Declarations.
* Advanced **CSS** Selectors.
* Mastering **CSS**, Part 1: Styling Design Elements.

# MICS

Shortcut key:

Comment out a block of code: Ctrl + /

Align code lines: Alt + Shift + F

Use tab to autofill code:

h1<Tab> 🡪 <h1></h1>

p>lorem100<Tab> 🡪 <p>lorem100</p>

Ctrl + Space for suggestion and one more time for further suggestion

#wrapper>#header+#footer<Tab>

1. 2 types of HTML elements: Block vs Inline
   1. Block Elements: A block element (such as <p>, <h1> to <h6> and <div>) starts on a new line, takes the full width, and ends with a new line. It is rectangular in shape with a line-break before and after the element.
   2. Inline Elements (or Character Elements): An inline element (such as <em>, <strong>, <code> and <span>) takes up as much space as it needs. It does not force a line-break before and after the element, although it can span a few lines.

In brief, a block element is always rectangular in shape, while an inline element spans a continuous run of characters.

1. Headings helps Google to understand the importance of your website’s content.
2. Image tag:

<img src=”path/to/img” alt=”message when the file is not loaded” title=””>  
title will show up when you hover your mouse on the image

1. Link to other pages (this is why web pages are called hypertext – texts linking to other texts)

<a href=”path/to/something”> the title of the link </a>  
<a href=”path/to/something” target=”blank”> the title of the link </a>  
Note the target=”blank”: open the link in a NEW tab.

href = hypertext reference

1. List
   1. Unordered list

<ul type=”square”>

<li> list item 1</li>

<li> list item 2</li>

</ul>

* 1. Ordered list

<ol type="a" start="3"> <!—start at “c”, but have to use number instead of letter

        <li>Ann</li>

        <li>Bob</li>

        <li>Cindy</li>

</ol>

1. Span and div. HTML tags often apply some meaning to the content, like <p> (paragraph), <h1> (heading). The purpose of span tag and div tag are to group HTML chunks into “class” or “id” CSS selectors.

Span is for supplementary info like number of comments, quantity, price of a product.

1. Table:

<table> and then <tr> (row) and <td> (data – cell)

One can set 3 parts: head, body and foot <thead>, <tbody>, <tfoot>

In order to merge columns, rows, one use colspan or rowspan

1. Id vs name attributes of html tag

Use name attributes for form controls (such as <input> and <select>), as that's the identifier used in the POST or GET call that happens on form submission.

Use id attributes whenever you need to address a particular HTML element with CSS, JavaScript or [a fragment identifier](http://en.wikipedia.org/wiki/Fragment_identifier). It's possible to look up elements by name, too, but it's [simpler and more reliable](https://stackoverflow.com/questions/6351570/what-is-the-difference-between-javascripts-getelementbyid-and-getelementsbyna) to look them up by ID.

The name attribute is used when sending data in a form submission. Different controls respond differently. For example, you may have several radio buttons with different id attributes, but the same name. When submitted, there is just the one value in the response - the radio button you selected.

1. Label is often associated with an id so that the autofocus function is carried out  
   <label for=”passwordID”>Password: </label>

<input type=”password” id=”passwordID”>

1. Input tag contains many types: text, password, email, number, date, radio, checkbox, and most important submit. Because of many types, Input always start by: <input type=”…”>
2. Select box: Select with option of value 0 🡪 the option shown on the screen

<select>

<option value=”0”>select a city</option>

<option value=”1”>Phoenix</option>

</select>

1. Radio, Checkbox: radio/checkbox buttons of a same group must have same name. Radio/checkbox buttons should have values for submitting to server.

<input type=”radio” name=”Gender” value=”male”>

<input type=”radio” name=”Gender” value=”female”>

<input type=”checkbox” name=”Hobby” value=”Fishing”>

<input type=”checkbox” name=”Hobby” value=”Chess”>

1. Type = “textarea” is for text of more than 1 line while type=”text” for text of 1 line

textarea is not an input tag

<input type=”text” placeholder=”type something here”>

<textarea cols=”” rows=””> type something here </textarea>

1. Submit to submit all the data (both in input tag <input type=””> and non-input tags, like textarea, select) in the <form> tag containing the submit button

Variables submitted to servers have names defined by “name=”…””

There are 2 ways to submit:

* Input type=”submit”: Note that the value=”” is just the text on the submit button, not the value submitted to the server. Those values are the value in tags in <form> tag

<input type=”submit” value=”text on the submit button”>

* Button type=”submit”. Button allows non-text label on the button while input type=submit allows only tẽt

1. Hidden input. When a programmer wants submit info that is not entered by users, he can put it in a hidden input.

<input type=”hidden” name=”variable\_name” value=”valuetosubmit”>

1. A html page should be divided into divisions, e.g. “header”, “content”, “footer”. Use <div id=”header”> or <div id=”content”> to do this.

Since <div id=”header”> is so common that it becomes <header> in HTML5.

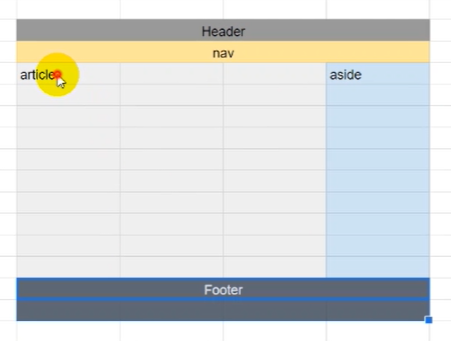
Note <header> is different from <head>.

1. Websites often have a dedicated section for navigational links (like Homepage, Back, Forward, etc). Since this is so common that this section becomes a tag <nav>. This tag is often put in <Header>
2. Article tag specifies independent, self-contained content. An article should make sense on its own and it should be possible to distribute it independently from the rest of the site.
3. <div> Vs <Section>

**<div>:** The [HTML](http://en.wikipedia.org/wiki/HTML) element (or HTML Document Division Element) is the generic container for flow content, which does not inherently represent anything. It can be used to group elements for styling purposes (using the class or id attributes), or because they share attribute values, such as lang. It should be used only when no other semantic element (such as <article> or <nav>) is appropriate.

**<section>:** The [HTML](http://en.wikipedia.org/wiki/HTML) Section element (<section>) represents a generic section of a document, i.e., a thematic grouping of content, typically with a heading. For exp: each chapter should be a section.

1. Aside is used for example ads
2. A structure of a webpage: header, nav, body (articles), footer, aside



# CSS

1. 3 ways to insert CSS with Keyword = Style

Inline CSS: style=”color: red;”

Internal CSS:

<style>

h1{color: red;}

</style>

External CSS:

<style>

<link rel=”stylesheet href=”…”>

</style>

1. 3 types of selectors with specificity increasing order: element, class, id (Note an id is always unique)

Another type of selector: attribute

An exp of Id selectors is footer.

element{

}

.class{

}

#id{

}

[attribute]{}

1. Why redundance makes sense

Exp:

ul#mylist{

}

#mylist is enough to identify which element but one still writes ul#mylist in order to show where mylist stays at. It’s in mylist.

1. Common IDs are: header, footer, sidebar

Common CLASSes are: external-link, highlight

1. Priority of CSS:
   1. Inline > Internal > External
   2. Selectors for specific elements > selectors for general elements. Exp: ID > CLASS
   3. In the CSS file, selectors below will override selectors above

!important increase the priority one level

1. CSS unit

Absolute measure:

Px: pixel (=1/96 in)

Pt: point (=1/72 in)

(note that the size of a pixel is the same for any device. The DPI (dot per in) is actually PPI (pixel per in))

The default size for fonts in website is 16px.

Relative measure:

Em: Relative to the font-size of the current element.

E.g. if font-size of a tag is set 20px and then for a child element you set 2em, it will be 40px.

Rem: Relative to the font-size of the ROOT element.

I guess Root element is the default font-size of browsers, which is 16px.

line-height: 1.5 means 1.5\*font-size

Properties of text-decoration:

overline, underline, line-through, none

The property none is used to remove the underline for <a> tag that by default has underlines

**Section 5. properties for div**

82.

width, height: The width and height of a div, which can be used as a box

The width, height, by default, is set to automatically fit the content (text, images) inside.

When the content is smaller than the min-width, min-height, then min-width, min-height are applied

* Better to set only min-width, min-height

text-align: center, justify, left, right

margin:

4 values: 20px 30px 40px 24px 🡪 top, right, bottom, left

3 values: 20px 30px 40px 🡪 top, left and right, bottom

2 values: 20px 40p 🡪 top and bottom, left and right

auto: to center horizontal. Auto is used only when there are two values:

20px auto 🡪 top and bottom and horizontal center

83.

color: the font color

background-color: background color

84. 85.

background-image: url(“a link”) 🡪 set up an image at the link as the background

background-size: 300px 500px or auto (original size) or cover (stretch to cover) or contain (resize to fit) or 30% 60% (percentage of the parent element)

background-repeat: no-repeat, repeat-x (only one horizontal line), repeat-y (only one vertical line), repeat (= repeat-x and repeat-y)

background-position: If you only specify one keyword, the other value will be "center"

left top, left center, left bottom, right top, right center, right bottom, center top, center center, center bottom

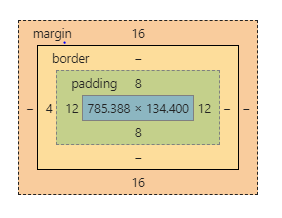
90. 91.

Margin – Border – Padding – Content

Margin 🡪 position of the div: space from the leftmost, rightmost, bottommost, topmost point

Border, Padding are additional parts to the Content they will make the div bigger.

box-sizing: content-box or border-box

the size of the box is set for content (and hence the real size will be bigger due to the padding and margin)

or border (and hence the real size will equal to the set value)

The width, height is the fixed size of a div; if none of them is indicated then the size will automatically fit the content.

**Every HTML element is either a block-level element or an in-line element.**

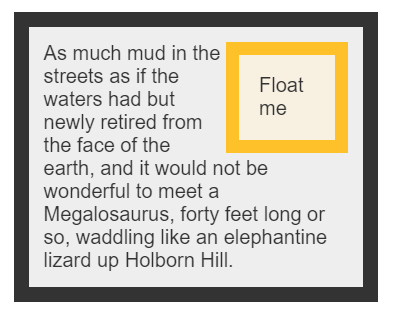
A block-level element consumes the whole line while an in-line element consumes only what it needs so there is space for other element to stand besides an in-line element.

Block elements: <p>, <h1>, <div>

In-line elements: <a>, <img>, <i>, <em>, <u>

**<div> is a block element and <span> is the in-line version of <div>**

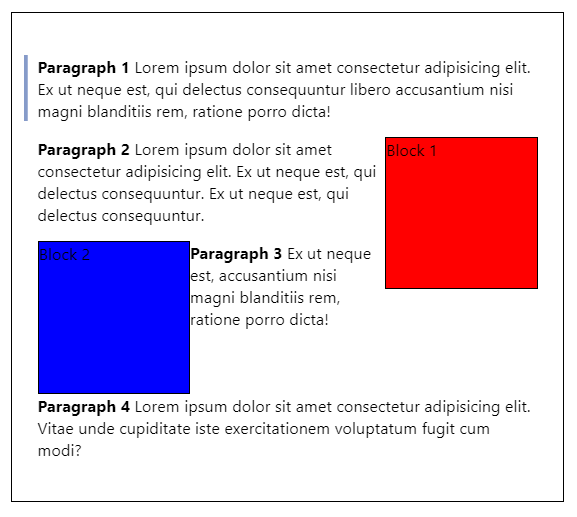
96. 97. Float and Clear

If an element is marked “float: left”, it will float on the top of the element below it on the left.

If an element is marked “clear: left”, it will prevent any element marked “float” from floating on it on the left.

Exp: the “float me” element is set “float: right”.

A div with “clear: left” property will not reside next to any “float” element 🡪 a new “line” will be created.

The order of those HTML elements:

Paragraph 1

Block 1 – float: right  
(Block 1 is on the top of Para 2 and it’s float: left)

Paragraph 2

Block 2 – float: left

(Block 2 is on the top of Para 3 and it’s float: right)

Paragraph 3

Paragraph 4 – clear: left  
(Para 4 is clear: left so block 2 and block 1 can’t jump on it)

98. **display**: inline, inline-block, block

When you create a heading, a new line is always created while when you mark some text italic there is no new line created. This is called “display: inline” property.

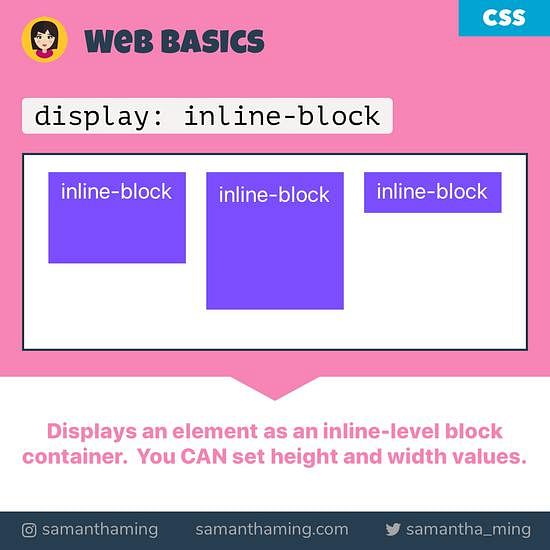
**Inline**: The element doesn’t start on a new line and only occupy just the width it needs. You can’t set the width or height.

**inline-block**: It’s formatted just like the inline element so it doesn’t start on a new line. BUT, you can set width and height values.

**Block**: The element will start on a new line and occupy the full width available. And you can set width and height values.

inline: elements with default “display: inline” property <i>, <em>, <u>, <img>, <a>

block: elements with default “display: block” property <p>, <h1>, <div>

FLEX BOX

101 102

Display: Flex

2 kind of elements of FLEX mechanism:

* Container, aka FLEX
* Items/boxes inside.

Both FLEX and Items inside are <div> elements. Not only FLEX, but also its items have properties relating to FLEX.

Properties for FLEX:

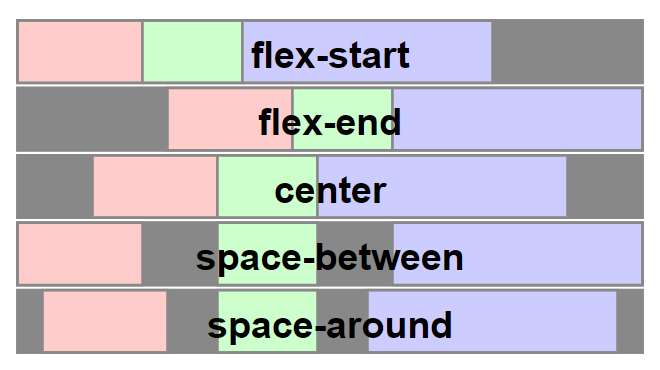
**display: flex**: this is to mark a flex container

**flex-wrap**: use “flex-wrap: wrap” if you want a new line. When the container is not set “flex-wrap: wrap”, boxes inside it will be compressed to fit the container regardless boxes’ size being set. When the boxes cannot be compressed any more, they will break the container. Using “flex-wrap: wrap” will create a new line when the flex runs out of space for its boxes.

**flex-direction**: row, column,

**justify-content** (align the content horizontally): center, space-between (adding space between items), space-around,

**align-content** (align the content vertically): center, space-between (adding space between items), space-around,



Properties for items:

**flex-basis**: similar to width. It can be auto. A flex-basis value set to auto sizes the element according to its size property (which can itself be the keyword auto, which sizes the element based on its contents).

**flex-grow**: Flex-grow is used for growing items to fill up the FLEX’s space. Flex-grow of items, e.g. 1 – 2 – 3 is the ratio for space for the items.

**flex-shrink**: flex-shrink is used for shrinking items when the screen is shrunk.

Those 3 properties can be written in a single line: flex: grow shrink basis.

**order**: the order of an item in a flex can be reassigned by using “order: …”. With this the order of the position in the HTML will be overridden.

**display**: inline, block. (see the “float” section above)

102. 103.

**“position: “** property goes with **top, left, right, bottom**

Exp:

  position: relative;  
  left: 30px;

**“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.

**“position: absolute”** (this always goes with **position: relative** of the parent) 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.

If you want to use “position: absolute” then you need to do 2 things: a parent element containing a child element and the both are set with “position: relative” and “position: absolute”.

**The normal position of an element is not fixed and hence comes “position: relative”**

**The position of the parent element is fixed and hence comes “position: absolute”**

**“position: fixed”** is used to fix a heading when you scroll the webpage.

10?

“**z-index: 3**” the order of layers

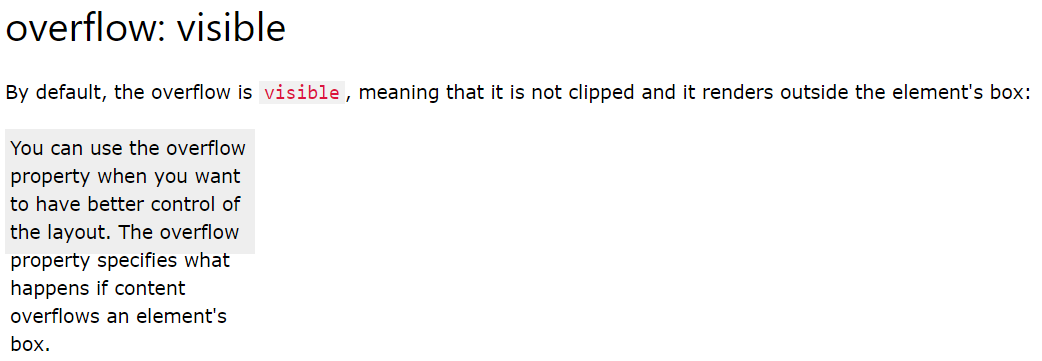
10?

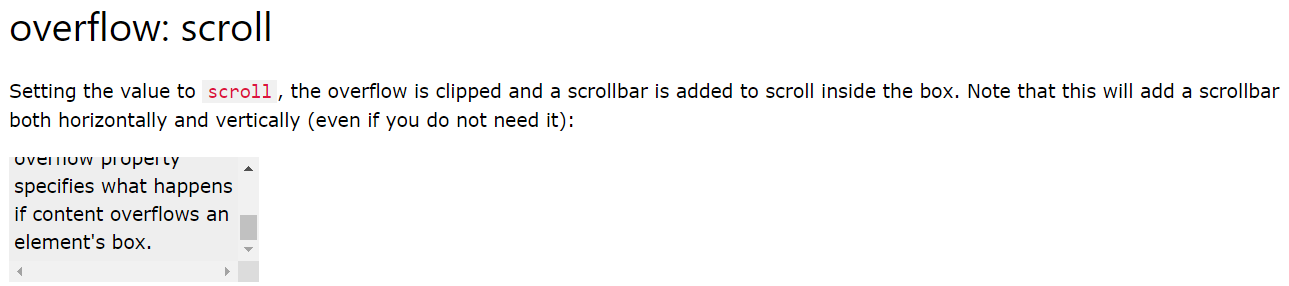
The **overflow** property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

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

**overflow**: auto or visible. This is used when the content doesn’t fit its container. The default value is visible.





119 transition

CSS transitions allows you to change property values smoothly, over a given duration.

To create a transition effect, you must specify two things:

* the CSS property you want to add an effect to
* the duration of the effect

Exp:

div {  
  transition: width 2s, height 4s;  
}

120 transform

transform: rotate(230deg)

transform: scale(x\_scale, y\_scale)

A note about height, width: 100% (fit parent) or auto (fit child)

height: 100% gives the element 100% height of its parent container.

height: auto means the element height will depend upon the height of its children.

Consider these examples:

**height: 100%**

<div style="height: 50px">

<div id="innerDiv" style="height: 100%">

</div>

</div>

#innerDiv is going to have height: 50px

**height: auto**

<div style="height: 50px">

<div id="innerDiv" style="height: auto">

<div id="evenInner" style="height: 10px">

</div>

</div>

</div>

#innerDiv is going to have height: 10px

# Most important about CSS

1. Margin – Border – Padding
2. Arrange elements in a line or not.

HTML elements can be

i) block: takes a whole line; its width/height can be modified. Exp: like <h1>, <p>, <div>

ii) in-line: take only the space it needs; its width/height cannot be modified. Exp: <i>, <emp>, <u>

<div> is a block and <span> is the in-line version of <div>

The property “display: in-line/block” will reset the in-line/block property of an element.

1. Float element: an element can “float: left” float on the left of the element below it and hence those two elements become in-line. But “float” is more flexible than “in-line” for exp: a tall float element can interfere two blocks (those two blocks are in two lines).
2. Position: set the position relatively to its parent or set it fixed
3. Width/height: 100% (fit parent), auto (fit child)
4. Overflow: when the content doesn’t fit its size.

# Javascript

## HTML DOM (Document Object Model)

### DOM allow Javascript to change content, structure, styles of HTML doc

Document Object Model represents every element in a HTML document as objects so that JavaScript can access and change the content, structures, styles of the HTML document.

Javascript can change, react, create, remove elements, attributes, styles.

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

### The HTML DOM Tree of Objects



### getElementById method and innerHTML property are used to change element

Example

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

## HTML with Javascript

Attributes for triggering something when an event occurs for any element (not only button, but also h1, img, etc): onclick, onmouseover, …

## Callback

A callback is a function passed as an argument to another function.

Exp: In the following, myDisplayer is the name of a function and it is passed to myCalculator() as an argument.

function myDisplayer(some) {  
  document.getElementById("demo").innerHTML = some;  
}  
  
function myCalculator(num1, num2, myCallback) {  
  let sum = num1 + num2;  
  myCallback(sum);  
}  
  
myCalculator(5, 5, myDisplayer);

When you pass a function as an argument, remember not to use parenthesis.

## Javascript asynchronous

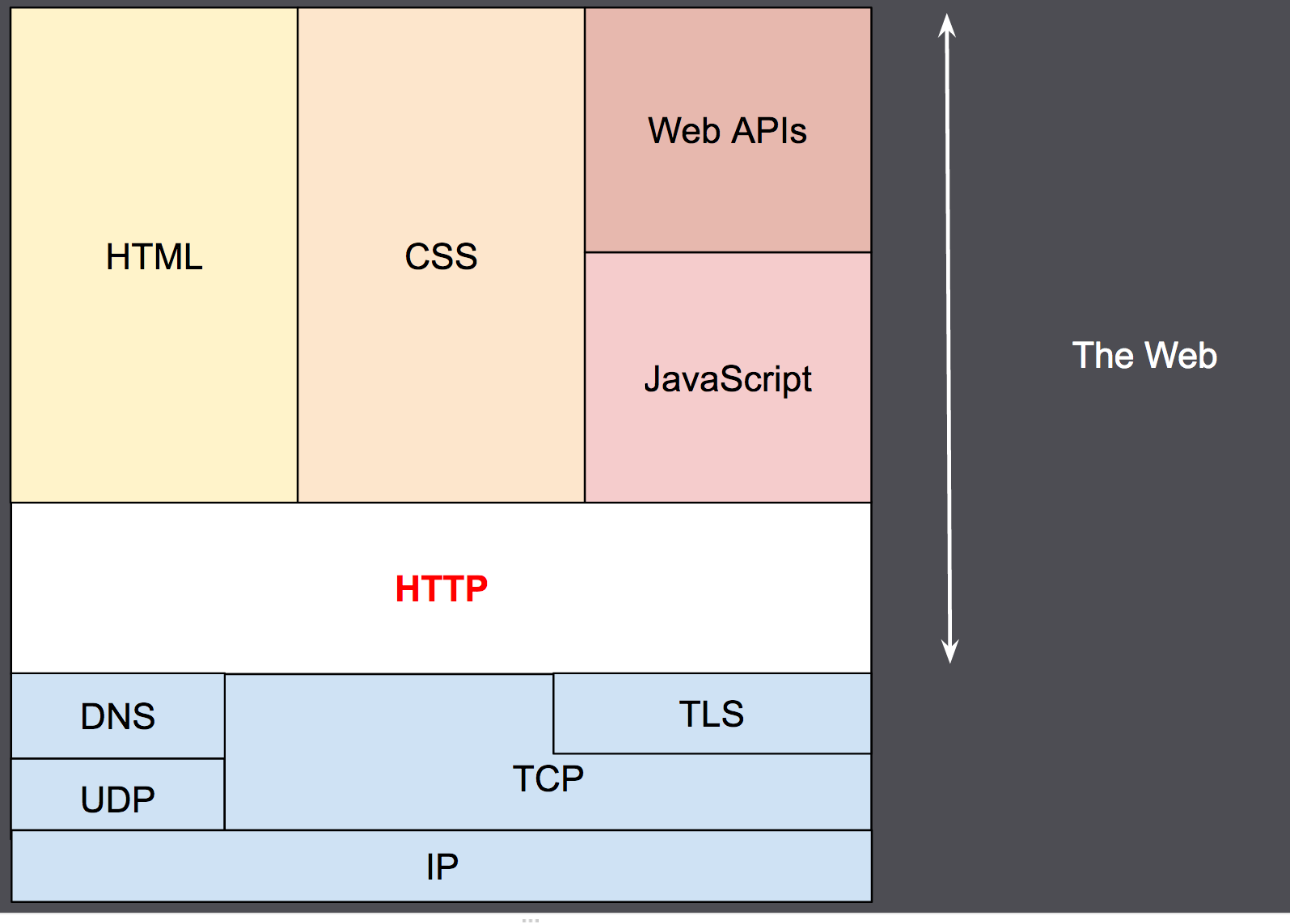
# HTTP

### HTTP is stateless and media independent

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

There are three basic features that make HTTP a simple but powerful protocol:

* HTTP is connectionless: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server knows about each other during current request and response only. Further requests are made on new connection as if client and server are new to each other.
* HTTP is media independent: It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.
* HTTP is stateless: As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.



### HTTP message

HTTP is based on the client-server architecture model and a stateless request/response protocol that operates by exchanging messages across a reliable TCP/IP connection.

HTTP requests/response consists of request/status line, header (field:value), message.

* **Start-line** (request/status)
* **Header**: “field:value” pairs (each is followed by a CRLF)
* An empty line indicating the end of the header fields
* **Message** (optional)

#### Start-Line

A start-line will have the following generic syntax:

start-line = Request-Line | Status-Line

Exp:

GET /hello.htm HTTP/1.1 (This is Request-Line sent by the client)

HTTP/1.1 200 OK (This is Status-Line sent by the server)

#### Header Fields

HTTP header fields provide required information about the request or response, or about the object sent in the message body. Each header field is of this form:

message-header = field-name ":" [ field-value ]

Exp:

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

Accept-Language: en, mi

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 51

Vary: Accept-Encoding

Content-Type: text/plain

#### Message Body

The message body part is optional for an HTTP message but if it is available, then it is used to carry the entity-body associated with the request or response. If entity body is associated, then usually **Content-Type** and **Content-Length** headers lines specify the nature of the body associated.

A message body is the one which carries the actual HTTP request data (including form data and uploaded, etc.) and HTTP response data from the server ( including files, images, etc.). Shown below is the simple content of a message body:

<html>

<body>

<h1>Hello, World!</h1>

</body>

</html>

### HTTP message example

#### Example 1

HTTP request to fetch **hello.htm** page from the web server running on tutorialspoint.com.

**Client request**

GET /hello.htm HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

**Server response**

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

Content-Length: 88

Content-Type: text/html

Connection: Closed

<html>

<body>

<h1>Hello, World!</h1>

</body>

</html>

#### Example 2

HTTP request to fetch **t.html** page that does not exist on the web server running on tutorialspoint.com.

**Client request**

GET /**t.html** HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

**Server response**

HTTP/1.1 404 Not Found

Date: Sun, 18 Oct 2012 10:36:20 GMT

Server: Apache/2.2.14 (Win32)

Content-Length: 230

Content-Type: text/html; charset=iso-8859-1

Connection: Closed

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html>

<head>

<title>404 Not Found</title>

</head>

<body>

<h1>Not Found</h1>

<p>The requested URL /t.html was not found on this server.</p>

</body>

</html>

#### Example 3

HTTP request to fetch **hello.htm** page from the web server running on tutorialspoint.com, but the request goes with an incorrect HTTP version:

**Client request**

GET /hello.htm **HTTP1**

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

**Server response**

HTTP/1.1 400 Bad Request

Date: Sun, 18 Oct 2012 10:36:20 GMT

Server: Apache/2.2.14 (Win32)

Content-Length: 230

Content-Type: text/html; charset=iso-8859-1

Connection: Closed

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html>

<head>

<title>400 Bad Request</title>

</head>

<body>

<h1>Bad Request</h1>

<p>Your browser sent a request that this server could not understand.<p>

<p>The request line contained invalid characters following the protocol string.<p>

</body>

</html>

#### Example 4

HTTP request to post form data to **process.cgi** CGI page on a web server running on tutorialspoint.com. The server returns the passed name after setting them as cookies:

**Client request**

POST /cgi-bin/process.cgi HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com

Content-Type: text/xml; charset=utf-8

Content-Length: 60

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

first=Zara&last=Ali

**Server response**

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Content-Length: 88

Set-Cookie: first=Zara,last=Ali;domain=tutorialspoint.com;Expires=Mon, 19-

Nov-2010 04:38:14 GMT;Path=/

Content-Type: text/html

Connection: Closed

<html>

<body>

<h1>Hello Zara Ali</h1>

</body>

</html>

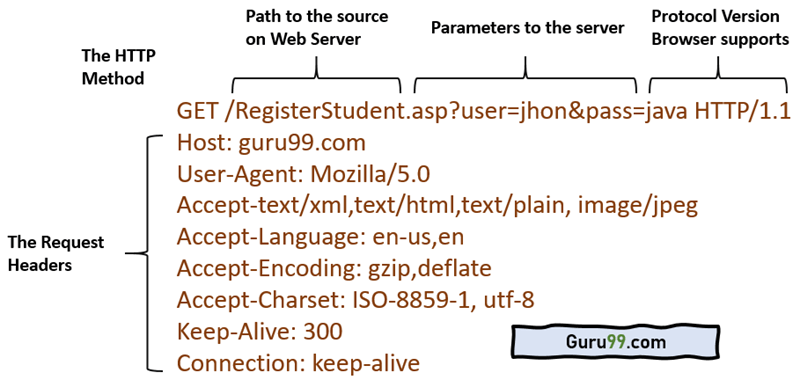
### GET vs POST

Methods of HTTP: GET, POST, PUT, HEAD, DELETE, PATCH, OPTIONS.

GET and POST are the most popular.

GET method is used to appends form data to the URL in name or value pair. Example:

GET/RegisterStudent.asp?user=value1&pass=value2



POST method requests that a web server accepts the data enclosed in the body of the request message, most likely for storing it.[[1]](https://en.wikipedia.org/wiki/POST_(HTTP)#cite_note-RFC_def-1) It is often used when uploading a file or when submitting a completed [web form](https://en.wikipedia.org/wiki/Form_(HTML)). Example:

POST/RegisterStudent.asp HTTP/1.1

Host: www.guru99.com

user=value1&pass=value2

