# 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>

# HTML

1. Structure of a html file

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

<!DOCTYPE html>

<html>

<head> </head>

<body> </body>

</html>

1. Comment: <!- -Comments -->
2. 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.

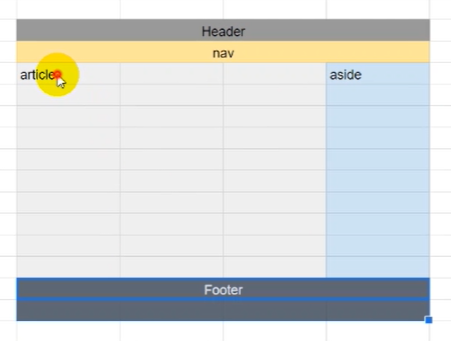
Note <header> is different from <head>.

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:

h1{color: red;}

</style>

External CSS:

<style>

<link rel=”stylesheet href=”…”>

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

## Basic selectors

[**Universal selector**](https://developer.mozilla.org/en-US/docs/Web/CSS/Universal_selectors)**:** \* will match all the elements

[**Type selector**](https://developer.mozilla.org/en-US/docs/Web/CSS/Type_selectors)**: element**

**Example:** input will match any [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) element.

[**Class selector**](https://developer.mozilla.org/en-US/docs/Web/CSS/Class_selectors)Selects all elements that have the given class attribute.  
**Example:** .index will match any element that has a class of "index".

[**ID selector**](https://developer.mozilla.org/en-US/docs/Web/CSS/ID_selectors)**:** Selects an element based on the value of its id attribute.

**Example:** #toc will match the element that has the ID "toc".

[**Attribute selector**](https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors)Selects all elements that have the given attribute.  
**Example:** [autoplay] will match all elements that have the autoplay attribute set (to any value).

## Grouping selectors

[**Selector list**](https://developer.mozilla.org/en-US/docs/Web/CSS/Selector_list)**:** The , is a grouping method, it selects all the matching nodes.  
**Example:** div, span will match both [<span>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span) and [<div>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div) elements.

## Combinators

[**Descendant combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/Descendant_combinator)The   (space) combinator selects nodes that are descendants of the first element.  
**Example:** div span will match all [<span>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span) elements that are inside a [<div>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div) element.

[**Child combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/Child_combinator)The > combinator selects nodes that are direct children of the first element.  
**Example:** ul > li will match all [<li>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/li) elements that are nested directly inside a [<ul>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ul) element.

[**General sibling combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/General_sibling_combinator)The ~ combinator selects siblings. This means that the second element follows the first (though not necessarily immediately), and both share the same parent.  
**Example:** p ~ span will match all [<span>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span) elements that follow a [<p>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p), immediately or not.

[**Adjacent sibling combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/Adjacent_sibling_combinator)The + combinator selects adjacent siblings. This means that the second element directly follows the first, and both share the same parent.  
**Example:** h2 + p will match all [<p>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p) elements that directly follow an [<h2>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/h2).

[**Column combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/Column_combinator)The || combinator selects nodes which belong to a column.  
**Example:** col || td will match all [<td>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/td) elements that belong to the scope of the [<col>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/col).

**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

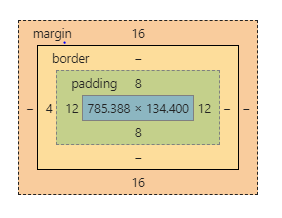
90. 91.

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.**

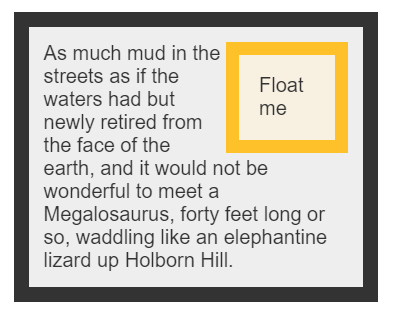
A block-level element consumes the whole line while an in-line element consumes only what it needs so there is space for other element to stand besides an in-line element.

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>**

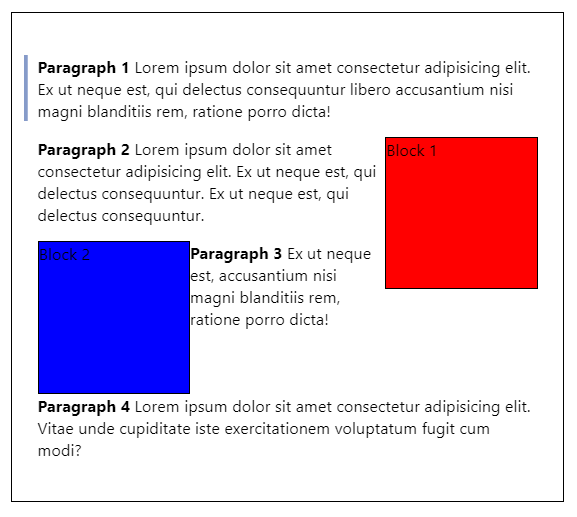
96. 97. Float and Clear

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.

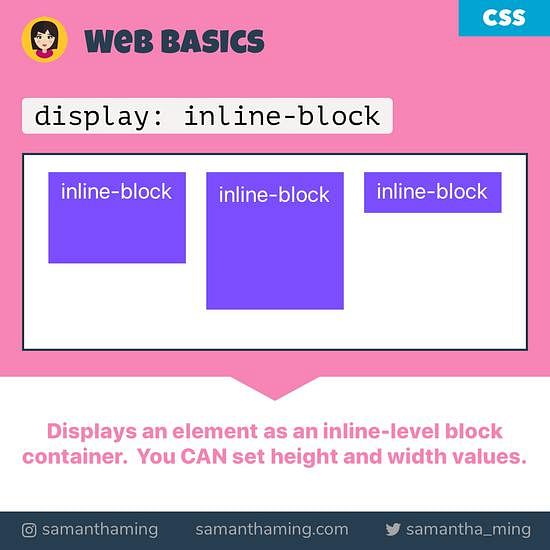
**Inline**: The element doesn’t start on a new line and only occupy just the width it needs. You can’t set the width or height.

**inline-block**: It’s formatted just like the inline element, where 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:

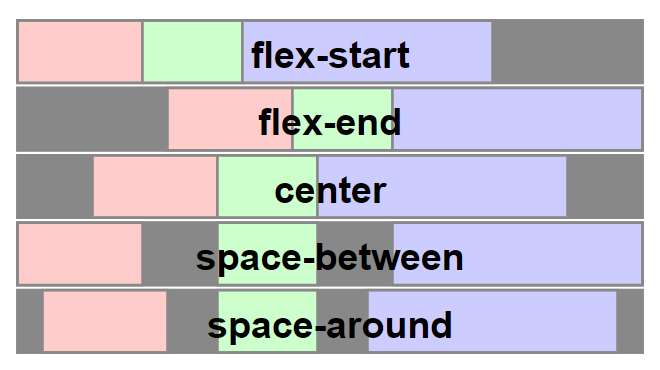
**display: flex**: this is to mark a flex container

**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

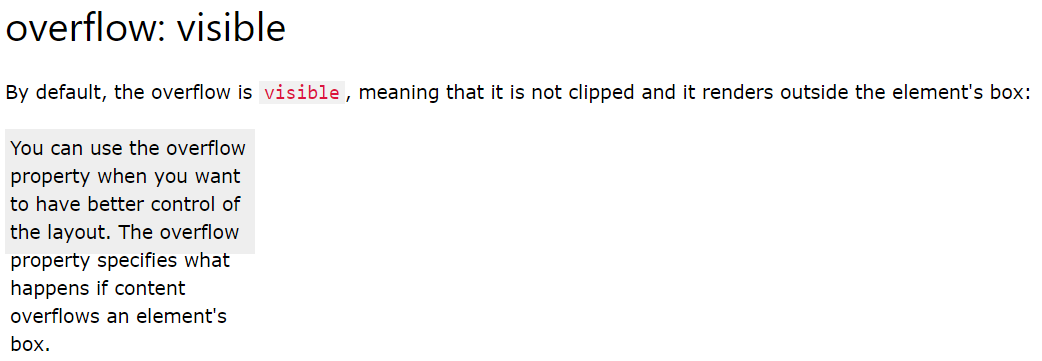
10?

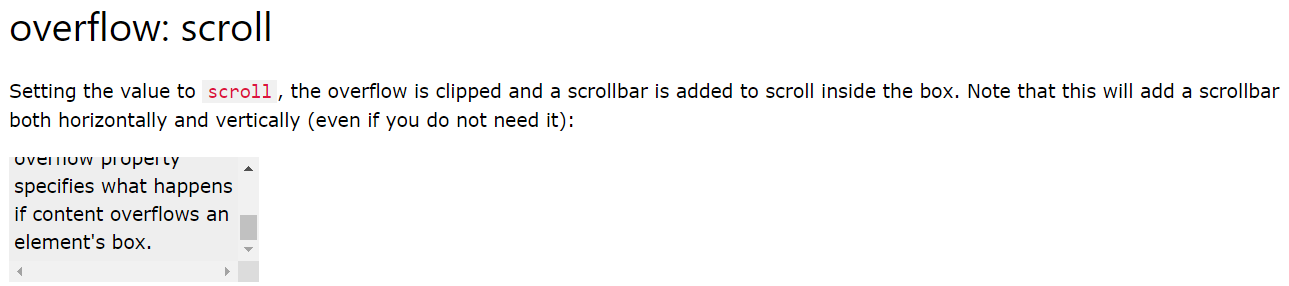
The **overflow** property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

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 and HTTP responses consists of the following four items.

* A Start-line
* Zero or more header fields followed by CRLF
* An empty line (i.e., a line with nothing preceding the CRLF)indicating the end of the header fields
* Optionally a message-body

#### Message 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>

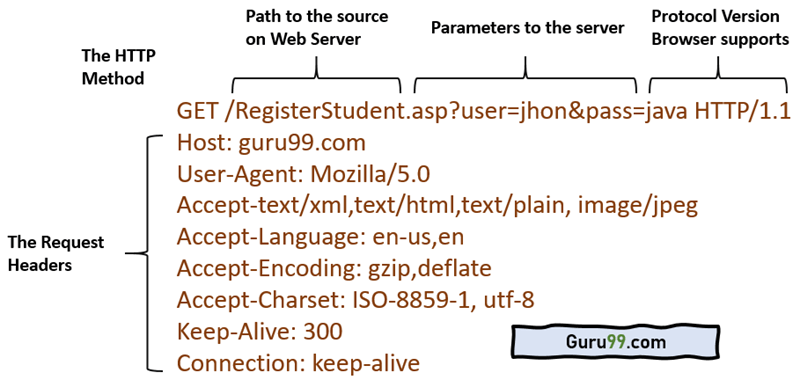
### GET vs POST

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

