Web Design

A Beginner's Handbook to HTML5 and CSS3

By Sandesh Paudel

Contents:

Industry Standards

❖ **Bullet-Proof Websites:**
-follow **industry standards** and **industry best-practices**

-features of a bullet-proof website:

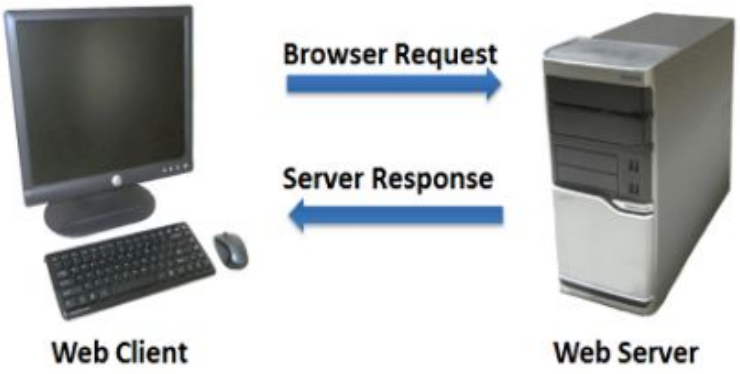
Updateability	F. the content is structured in a way so it can be managed easily
Compatibility	C. the web site will work with all web browsers, now and in the future
Reusability	E. the code can be developed quickly and efficiently
Dependability	B. the content doesn't have mistakes
Manageability/Flexibility	A. the website can grow and evolve with the business
Consistency	G. the website has a consistent look and feel
Find-ability	D. the website can be found easily in search engines

❖ **Basics of a website:**
- **Domain Name:**
  > **ICANN** [Internet Corporation for Assigned Names and Numbers] is responsible for coordinating the assignment of domain names.
  > **Registration:** In order to register a domain name, you would get a **third party company** [like godaddy.com] who in turn contacts ICANN for you.

- **Web Server:**
  > a computer with a connection to the internet, running web server software like Apache.

- **Web Page Files**

History and Definitions



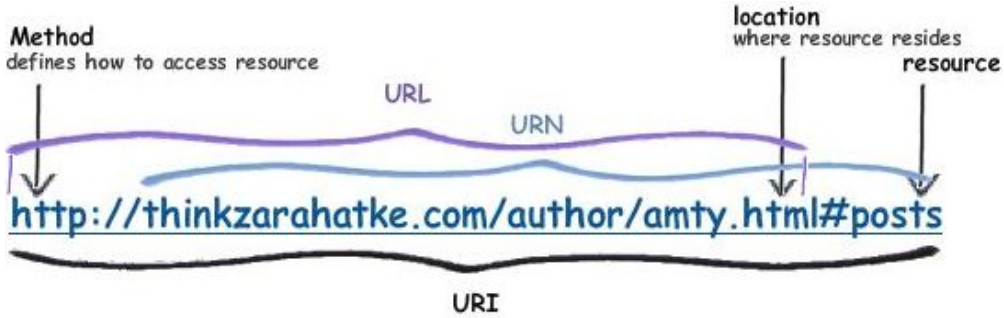
- **The Internet != Web:**
  -Contrary to popular belief, the Internet is not the same thing as web. The **internet is a network** such as Email [1971] , Telnet [1972], FTP [1973], Gopher [1991], etc and can be command-line based (mostly).

  -Web is a subset of the internet that uses HTTP protocol [Web Pages + Hyperlinks]

  -Some common Network Models/topologies:
  - Peer-to-peer
  - Star
  - Hierarchical
  - Client/Server ← important (Internet)

- **TCP/IP:**
  -The TCP [**Transmission Control Protocol**] can be thought of as the envelope. The web browser to the TCP layer, which chops up the data into bits [envelopes], and sends them to the IP address. IP [**Internet Protocol**] can be thought of as the address on that envelope.

- **URI vs URL:**
  > **URI:** Uniform resource Identifier [start.html]
  > **URL:** Uniform Resource Locator [http://google.com]



❖ **History:**

> **1991 -** Tim Berners Lee introduced the web in 1991. He combined three technologies: documents on the **internet, markup languages** [xml, xhtml, html5, etc for syntactical notation], and **hypertext** [text that contains links to other texts]. The original purpose of the WWW was to share research papers between scientists.

> **1993 -** Marc Anderson creas the first graphical web browser that could display inline images. It was called "Mosaic".

Mosaic → Netscape → Mozilla → Firefox

❖ **Progressive Enhancement for Web Development:**

- the general strategy for structuring web development in the following layers:

1. Structure
   - HTML - hypertext markup language
   - Proper tags enable the "worldwide database" ...big data
2. Presentation
   - CSS - cascading style sheets (next week)
   - formatting and layout
   - E.g. red = danger
3. Behavior
   - JavaScript (and others)
   - User interactions (clicking, tapping - things move around on the screen)

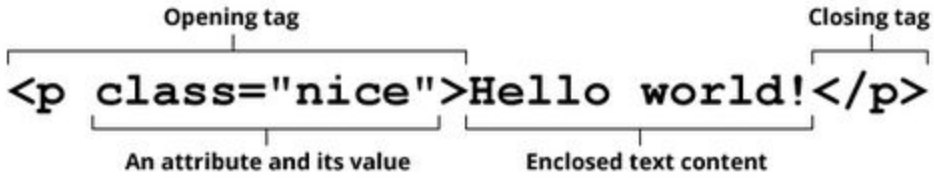
HTML Document Structure

```
<!DOCTYPE html> [1]

<html lang="en"> [2]

  <head>

    <meta charset="utf-8"> [3]
    <title>Page Title</title>
    <meta name="description" content="Roughly 155
    characters">

    <link rel="stylesheet" type="text/css" href="
    mystyle.css">

    <script src="https://ajax.googleapis.com/ajax/
    libs/jquery/1.7.1/jquery.min.js"></script>
    <script src="script.js"></script>


  </head>

  <body>
    <!-- Content -->
  </body>
</html>
```

1. **Document Type Declaration (DTD):** <!doctype html> indicates HTML5.
2. **HTML5 requirement:** <html lang= “en”>
3. **Meta Element:** provides information about the character-encoding.

❖ **Tags/Elements:**

Anatomy of an HTML element



❖ **Linking:**

> **Absolute Path:** Link

> **Relative Path:** Link

- Same directory

```
<a href="index.html">Undergraduates</a>
```

- Child

```
<a href="services/index.html">Undergraduates</a>
```

- Parent

```
<a href="../index.html">Home page</a>
```

- Sibling

```
<a href="../services/file.html">Home page</a>
```

❖ **Semantics:** aim to structure your HTML to convey the meaning of the content over appearance. This is helpful for SEO to index your page when proper semantics is used (for eg. using **** instead of **<b>**).

Use tags based on their meaning

- <p> means one paragraph - sentences!
- means emphasis
- means important

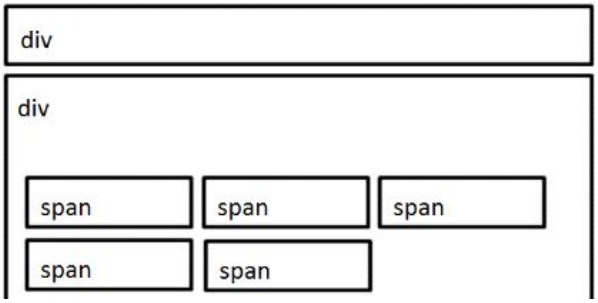
Catch-all (meaningless) tags

-
-

❖ **Structural Elements**

> **Block elements:** always start on a new line and takes up the full width. It is as tall as the content needs to be.
[for eg. **div, p, h[1-6], ol, ul, blockquote, form, table**]

> **Inline elements:** does not start on a new line. It lines up side by side, and is as wide as it needs to be. It is as tall as one line.
[for eg. **span, b, a, strong, em, img**]

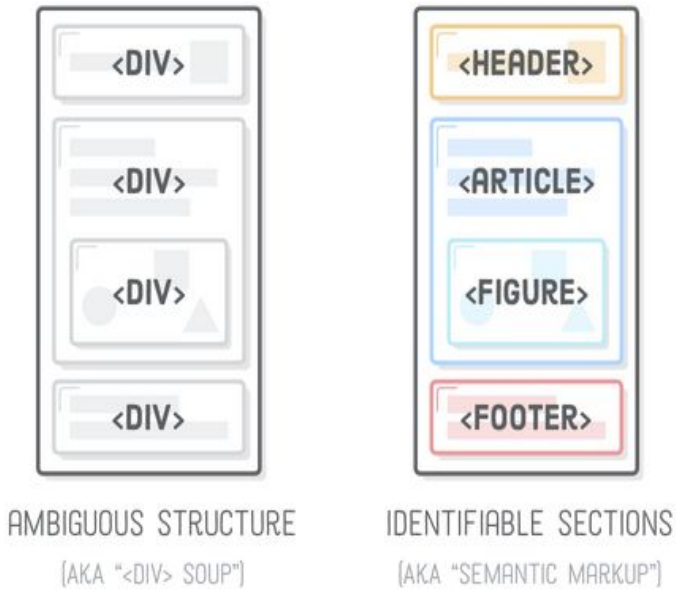


❖ **The deal with div and span**

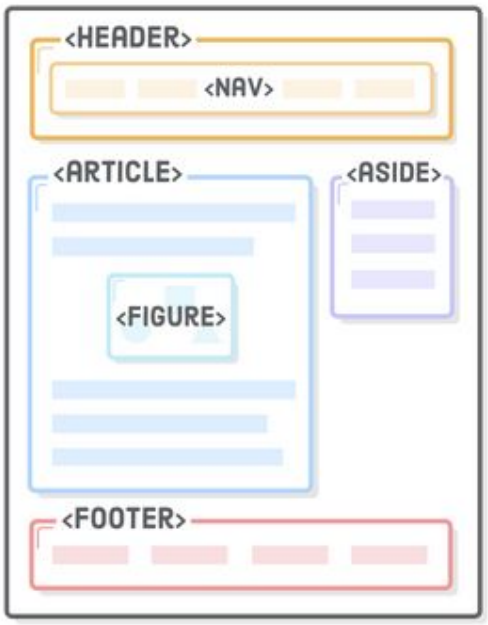
> **Old- fashioned**

> **div** is often used as a container to style its content with CSS or to perform certain tasks with JS.

> **div** and **span** have no semantic meaning at all.



❖ **Sectioning with HTML5**

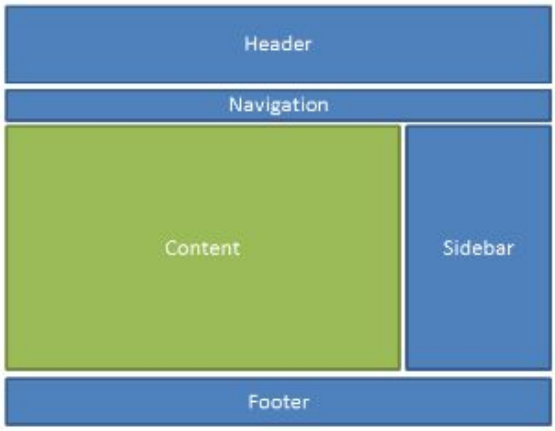


Example of a semantically valid code structure:

```
<body>

    <header>
        <h1>Daniel Radcliffe</h1>
        <div class="subtitle">The guy who starred as Harry Potter</div>
    </header>


    <nav class="menu">
    <ul>
        <li><a href="about.html" class="is-current">About</a></li>
        <li><a href="life.html">Life</a></li>
        <li><a href="religion.html">Religion</a></li>
        <li><a href="work.html">Work</a></li>
    </ul>
    </nav>


    <article>
        <h2> About Radcliffe </h2>

        <figure>
            <img src="face.png" alt="image1">
            <figcaption>Danel Radcliffe talking </figcaption>
        </figure>

        <p>Daniel Jacob Radcliffe (born 23 July 1989) is an English...</p>
    </article>


    <aside>

        <h2>More Information</h2>

        <p>Sources say, He has contributed to many charities, including Dem
        Children, and The Trevor Project for suicide prevention among LGBTQ
        Hero Award in 2011."</p>
    </aside>

</body>
```

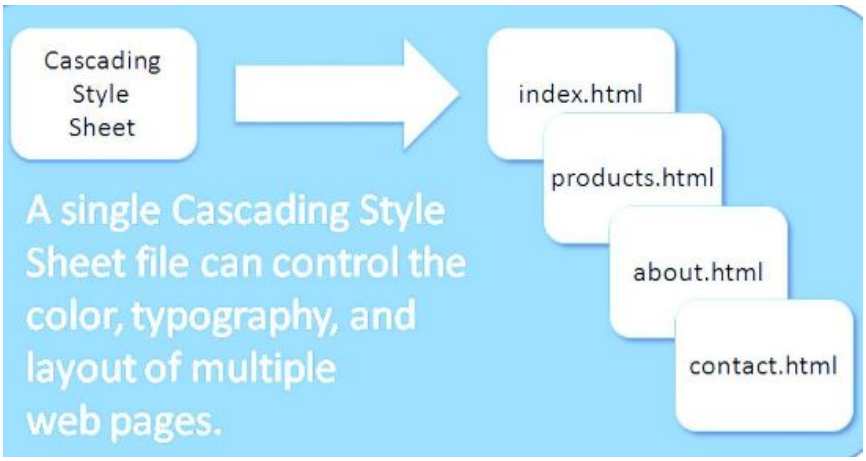
❖ **HTML Validator:** https://validator.w3.org/

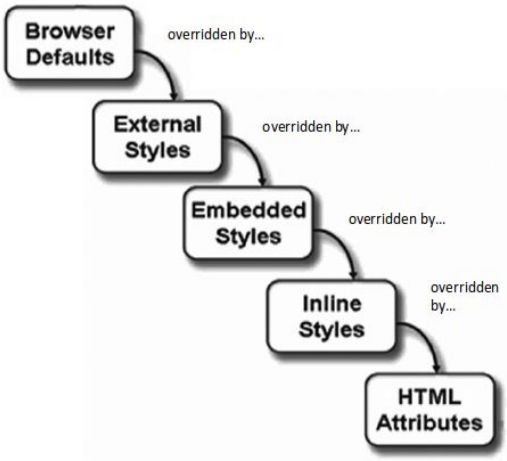
CSS3 Standards[2018]



- **Developed by W3C**: W3C creates recommendations and prototypes for web browsers and related web technologies.

- **CSS Uses**
  - > **Formatting** : font, colors, backgrounds, borders, etc.
  - > **Layout:** positioning and columnar layout.
  - > **Navigation**: menu formatting.





Include external css file

```
<link rel="stylesheet" type="text/css" href="/style.css" />
```

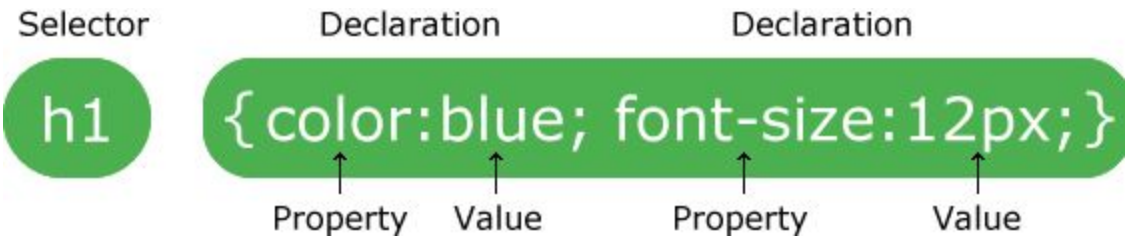
Internal styles

```
<style type="text/css">
  div { color: #444;}
</style>
```

Inline styles

```
<tag style="property: value"> </tag>
```

❖ **CSS Syntax:**



❖ **Selectors:**

Selectors

*	*all elements*
div	*all div tags*
div,p	*all divs and paragraphs*
div p	*paragraphs inside divs*
div > p	*all p tags, one level deep in div*
div + p	*p tags immediately after div*
div ~ p	*p tags preceded by div*
.classname	*all elements with class*
#idname	*element with ID*
div.classname	*divs with certain classname*
div#idname	*div with certain ID*
#idname *	*all elements inside #idname*

Pseudo classes

a:link	*link in normal state*
a:active	*link in clicked state*
a:hover	*link with mouse over it*
a:visited	*visited link*

element selector: p { }

class selector: .loud { }

id selector: #id { }

descendant selector: header p { }
.main-menu ul { }

group selector: .lead, h1, h2 { }

compound selector: p.lead { }

Pseudo-class	Purpose
:first-of-type	Applies to the first element of the specified type
:first-child	Applies to the first child of an element (CSS2 selector)
:last-of-type	Applies to the last element of the specified type
:last-child	Applies to the last child of an element
:nth-of-type(n)	Applies to the "nth" element of the specified type Values: a number, odd, or even

[1] BASIC FORMATTING [Colors, Background, Borders, Font, Text]

❖ Colors:

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

> **RGB and RGBA** [where A is transparency [1.0 = fully opaque]]:

rgb(255, 99, 71)

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

rgba(255, 99, 71, 0.5)

> **HEX [rrggbb]:**

#3cb371

❖ Background:

`background-color`
`background-image`
`background-repeat`
`background-attachment`
`background-position`

```
body {
    background: #ffffff url("img_tree.png") no-repeat right
top;
}
```

❖ Borders

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

```
p {
    border: 5px solid red;
}
```

> **Rounded Borders:**

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

Round border

❖ **Font:**

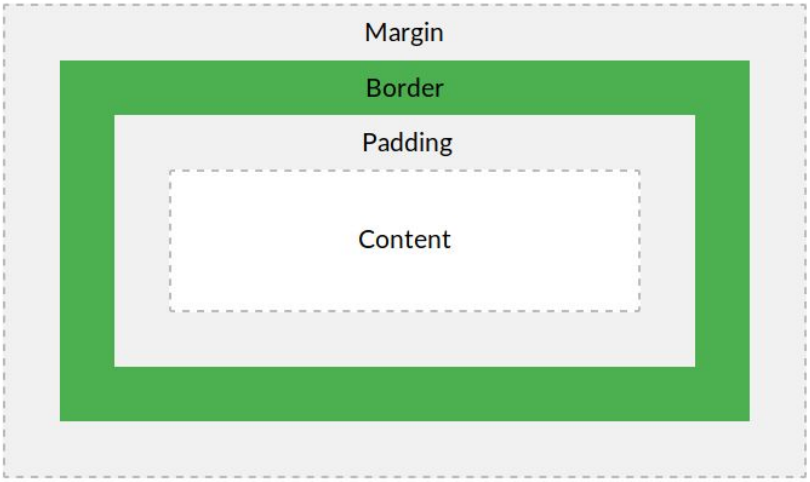
```
font-family: "Times New Roman", Times, serif;
font-style: normal; font-style: italic;
font-size: 2.5em; /* 40px/16=2.5em */
font-weight: bold;
```

❖ **Text:**

```
color: blue;
text-align: center;
text-decoration: none; // remove decorations from text
text-transform: uppercase;
text-indent: 50px; letter-spacing: 3px; line-height: 0.8;
word-spacing: 10px; text-shadow: 3px 2px red;
```

[2] POSITIONING [Box Model, Margin/Padding, Position Attribute, Float/Clear, Overflow, Sizing/Autoresize]

❖ The Box Model:




```
div {
    width: 320px;
    padding: 10px;
    border: 5px solid gray;
    margin: 0;
}
```

❖ Margin / Padding:

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

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

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

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

❖ Position Attribute

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

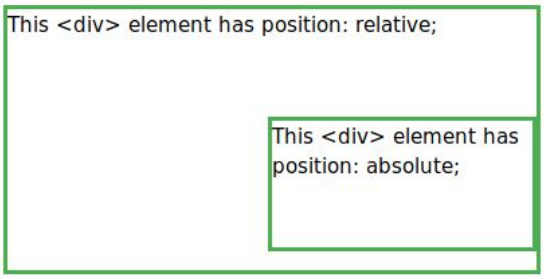
> **static:**
-by default
-position according to the normal flow of the page
-not affected by top, bottom, left, right properties

> **relative:**
-setting the top, right, bottom, left properties will cause it to be adjusted relative to its normal position. [creates a gap]

```
div.relative {
    position: relative;
    left: 30px;
    border: 3px solid #73AD21;
}
```

> **absolute:**
- positioned relative to the nearest positioned ancestor.

```
div.relative {
    position: relative;
    width: 400px;
    height: 200px;
    border: 3px solid #73AD21;
}

div.absolute {
    position: absolute;
    top: 80px;
    right: 0;
    width: 200px;
    height: 100px;
    border: 3px solid #73AD21;
}
```

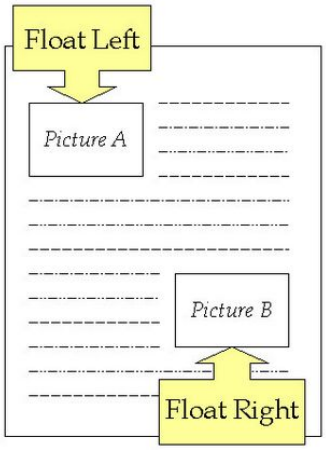


> **fixed:**
-positioned relative to viewport. Not affected by scrolling.

```
div.fixed {
    position: fixed;
    bottom: 0;
    right: 0;
    width: 300px;
    border: 3px solid #73AD21;
}
```

*there is also sticky, which is cool.

❖ **Float** [can be used for more than just images. Eg. Containers]



The `float` property can have one of the following values:

- left - The element floats to the left of its container
- right- The element floats to the right of its container
- none - The element does not float (will be displayed just where it occurs in the text). This is default
- inherit - The element inherits the float value of its parent

In its simplest use, the `float` property can be used to wrap text around images.

```
img {
    float: right;
}
```

> when using multiple floating items, it stacks side by side and resizes nicely:


```
<div class="after-box">
    I'm using clear so I don't float next to the above boxes.
```

Without clear

div1

div2 - Notice that div2 is after div1 in the HTML code. However, since div1 floats to the left, the text in div2 flows around div1.

With clear

.div3 {
float: left
};

.div4 { *clear: left* };
moves div4 down below the floating div3. The value "left" clears elements floated to the left. You can also clear "right" and "both".

❖ Overflow issues:

The `overflow` property specifies what should happen if content overflows an element's box.

This property specifies whether to clip content or to add scrollbars when an element's content is too big to fit in a specified area.

Note: The `overflow` property only works for block elements with a specified height.

The `overflow` property has the following values:

- `visible` - Default. The overflow is not clipped. It 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, but a scrollbar is added to see the rest of the content
- `auto` - If overflow is clipped, a scrollbar should be added to see the rest of the content

> Float Overflow ClearFix

here is the problem of using floated elements inside a div container:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...

> Overflow Clearfix:

.div{ *overflow : auto* }
Add a clearfix class with overflow: auto; to the containing element, to fix this problem:

❖ **Sizing - Height / Width:**
The height and width can be set to auto (this is default. Means that the browser calculates the height and width), or be specified in *length values*, like px, cm, etc., or **in percent (%) of the containing block**.

```
div {
    max-width: 500px;
    height: 100px;
    background-color: powderblue;
}
```

❖ **Autoresize:** use % to specify the height and the width proportional to the size of the parent block.

> **margin: auto;**

```
#main {
  width: 600px;
  margin: 0 auto;
}
```

Setting the width of a block-level element will prevent it from stretching out to the edges of its container to the left and right. Then, you can set the left and right margins to auto to horizontally center that element within its container. The element will take up the width you specify, then the remaining space will be split evenly between the two margins.

> **max-width:**
It is discouraged to use width because that causes problems when resizing the content. Use max-width to handle better resizing for smaller devices. It will resize up to the max-width for bigger displays, and scale down for smaller displays.

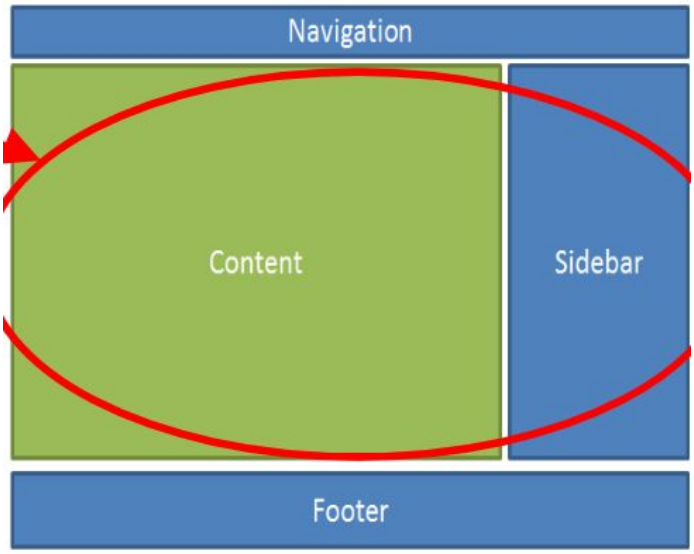
❖ **Object-Fit:** used to specify how <img> <video> should be resized to fit the container. [fill stretches the whole thing, contain contains it, etc] [fill| contain | cover | none | scale-down]

```
object-fit: fill;
```

[3] Layout: [Classic C-clamp, Navigation, Display, Layout Fundamentals. Flex, Grid, Grid+flex]

❖ **Common Layout Practices:**

> **columnar layout:** any layout where some of the content is positioned side by side.



> **Centering the page for the Classic "C-clamp":**

HTML

```
<body>
  <div class="container">
      Hello World!
  </div><!--.container-->
</body>
```

CSS

```
.container {
  margin: 0 auto;
  width: 90%;
  max-width: 960px;
}
```

❖ Navigation

1) Navigation Bar = List of Links

```html
<nav class="menu">
    <ul>
        <li><a href="about.html" class="is-current">About</a></li>
        <li><a href="life.html">Life</a></li>
        <li><a href="religion.html">Religion</a></li>
        <li><a href="work.html">Work</a></li>
    </ul>

</nav>
```

2) Style menu with [font, border, alignment]:

```css
.menu {
    padding-top: 30px;
    text-align: center;
}


.menu li {
    border: 2px solid #1C1C1C;
    font-family: "Courier New", Courier, monospace;
    border-radius: 5px;
    padding-left: 0px;
}
```

3) Remove the bullets, margins and padding [default list style]

```css
.menu ul {  /* undo default UL styles */
    margin: 0;
    padding-left: 0;
    list-style-type: none; /* typical - no bullets */
}
```

4) For horizontal nav bar [preferably use display inline over float]:


```
.menu li { display: inline-block; }
```

5) For Vertical nav bar:

```
.menu li { display:block; }
```

6) Other styles [clickable block area, padding, color, hover behavior, current page]

```
.menu a {
    display: block; /* typical - to make a larger clickable area
    padding: 10px 10px;  /* typical - style as needed */
    color: rgba(223, 1, 58, 0.6);
    font-weight: bold;

}

.menu a:hover {  /* typical - style as needed */
    color: #FFFFFF;
    background-color: rgba(223, 1, 58, 0.6);
}
.menu a.is-current {  /* required - style as needed */
    color: #FFFFFF;
    background-color: rgba(223, 1, 58, 0.6);
}
```

❖ Display

> Block elements:

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

[for eg. **div, p, h[1-6], ol, ul, blockquote, form, table**]

> Inline elements:

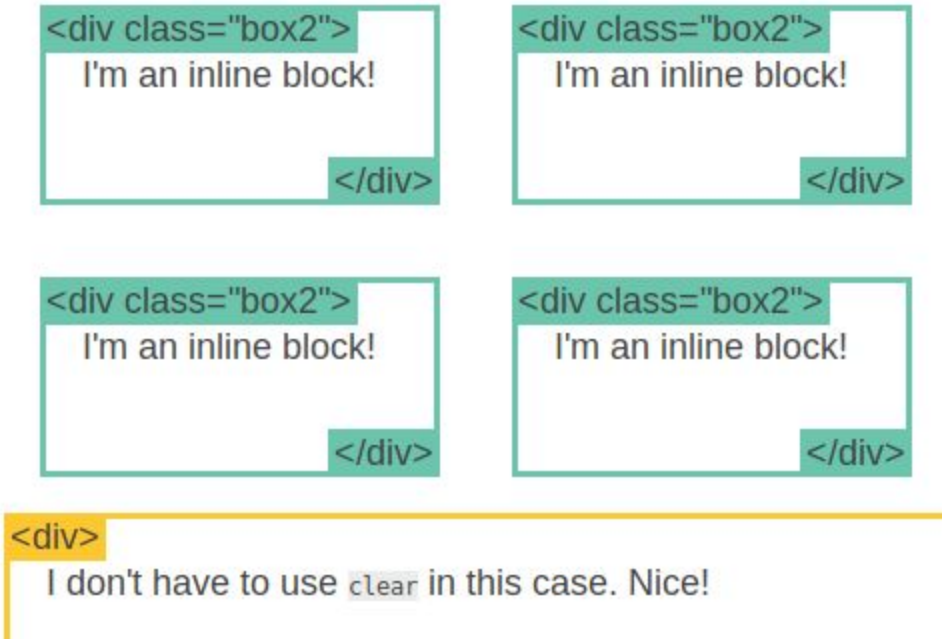
An inline element does not start on a new line and only takes up as much width as necessary. Cannot set height. Or Width.

[for eg. **span, b, a, strong, em, img**]

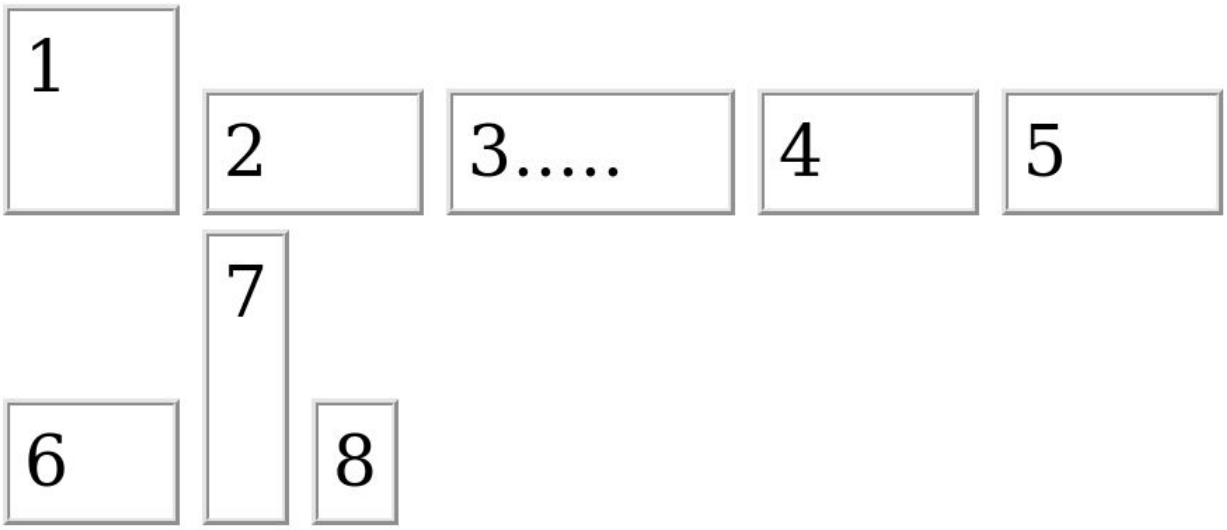
> CSS display property overrides the default html element display:

```
p.ex1 {display: none;}
p.ex2 {display: inline;}
p.ex3 {display: block;}
p.ex4 {display: inline-block;}
```

> More on inline-block: this has the same effect as float:left, but you do not have to use clear for the next set of block elements.



> arrangements of items with uneven heights with inline block display:



❖ **More Display:**

The CSS **display:** property

Defaults for all HTML elements that display content…

display: block;

- Stacks-up, top-over-bottom with other elements
- Uses the "box model" (padding, margin, borders)
- Is 100% wide (takes-up all the width on a line)

display: inline;

- Lines-up, side-by-side with other elements
- Ignores or does strange things to padding, margin, borders (no box model)
- Is only as wide as it needs to be; if wider than 100%, wraps to the next line

Options for layout…

display: inline-block;

- Uses the "box model" (like display: block) AND…
- Is only as wide as it needs to be (like display: inline)

display: table-cell;

- Neighboring block elements line-up side-by-side with equal height
- Ignores some aspects of the "box model"

display: flex;

- Neighboring block elements line-up side-by-side with equal height
- Uses the "box model"

display: grid;

- Everything within a GRID element can be placed anywhere you want, within the grid's box

❖ Layout Fundamentals

> **Layout blocks:** use <NAV> / <MAIN> / <SECTION> / <HEADER> / <ARTICLE>/ <ASIDE> / <FOOTER> / etc. Use Div container if you have to!

> **Layout techniques:**

- ~~HTML Table~~ (not acceptable for layout)
- ~~Float layout~~ (old fashioned but still widely used)
- Inline-block ← Best for small layouts (e.g. navigation elements)
- CSS table layout
- Flex layout

  } Current and best practices in the industry
- Grid layout ← New – complicated but powerful

> **Choices for layout:**

- **HTML tables** are for tabular data only! (Not for layout)
- The CSS **float** property is for moving small content to the side (left or right) and letting the rest of the content flow around it
- The CSS **inline-block** property is for lining up block elements side-by-side
- The CSS **table-cell** property is for creating a (simple) page layout with columns and rows (like a table, but not actually an HTML table)
- The CSS **flex** property is for lining up block elements side-by-side
- The CSS **grid** property is for creating a any page layout with columns and rows

❖ **Flex** [dynamically resizes!]

> **Purpose:** The Flexible Box Layout Module makes it easier to design flexible responsive layout structure without using float or positioning.


```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

```
.flex-container {
  display: flex;
}
```

> **Control Direction:**


```
.flex-container {
  display: flex;
  flex-direction: column;
}
```

```
flex-direction: column-reverse;
flex-direction: row;
flex-direction: row-reverse;
```

> **Wrapping:** to break on to the next line if necessary depending the size of the viewport.

```
flex-wrap: wrap; flex-wrap: nowrap; flex-wrap: wrap-reverse;
flex-flow: row wrap;
```

> **justify-content:** align the items within the flex container [center, flex-start, space-around, space-between, etc.]

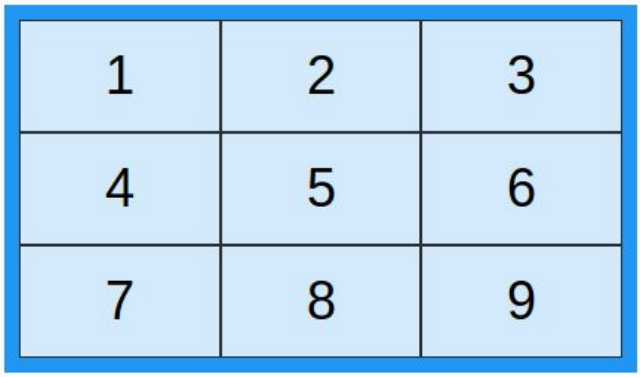
> **align-items:** align items vertically within the flex container.

> **many, many more properties. Look for them in the documentation.**

❖ **Grid** [dynamically resizes]

> **Purpose:** to offer a grid-based layout system with rows and columns, making it easier to design web pages without having to use floats and positioning.

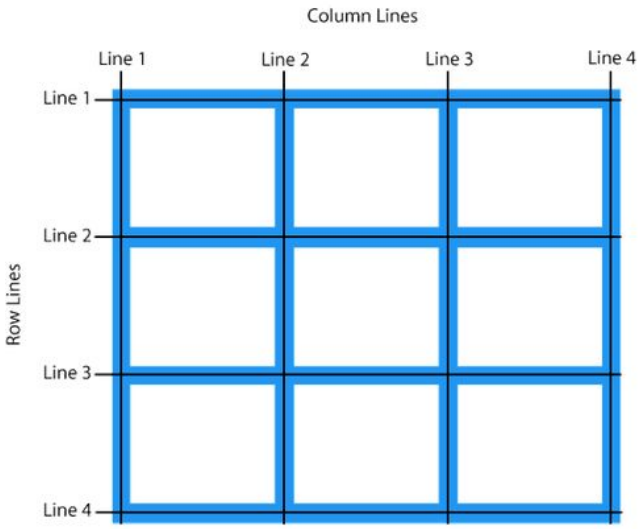
```html
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
  <div class="grid-item">7</div>
  <div class="grid-item">8</div>
  <div class="grid-item">9</div>
</div>
```



> **Grid containers have inline-grid or grid display property:**

```css
/**Container parent **/
.grid-container {

  display: grid;

  /**set up the number of rows and columns **/
  grid-template-columns: 20% 60% auto 1fr;
  grid-template-rows: auto auto auto auto;

  background-color: #2196F3;
  padding: 10px;
}
```

```css
/**Child Item **/
.grid-item1 {

    /** setting the palcement of the
    item in the grid **/

    grid-column: 1 / 2;
    grid-row: 1 / 4;

}
```



> **Row lines and column lines:**

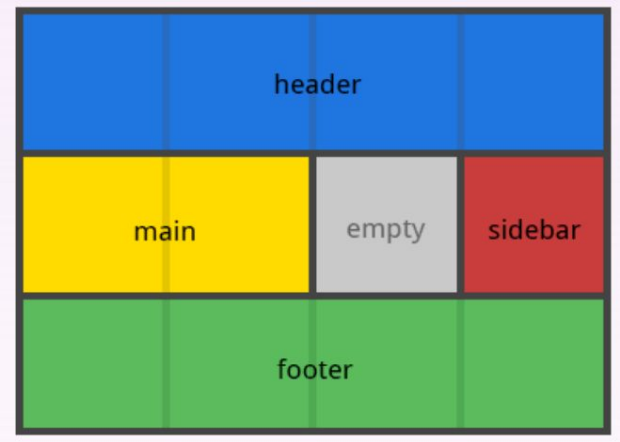
```
.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

```
.item1 {
  grid-row-start: 1;
  grid-row-end: 3;
}
```

> **Grid gaps** [shorthand for grid-row-gap and grid-column-gap]

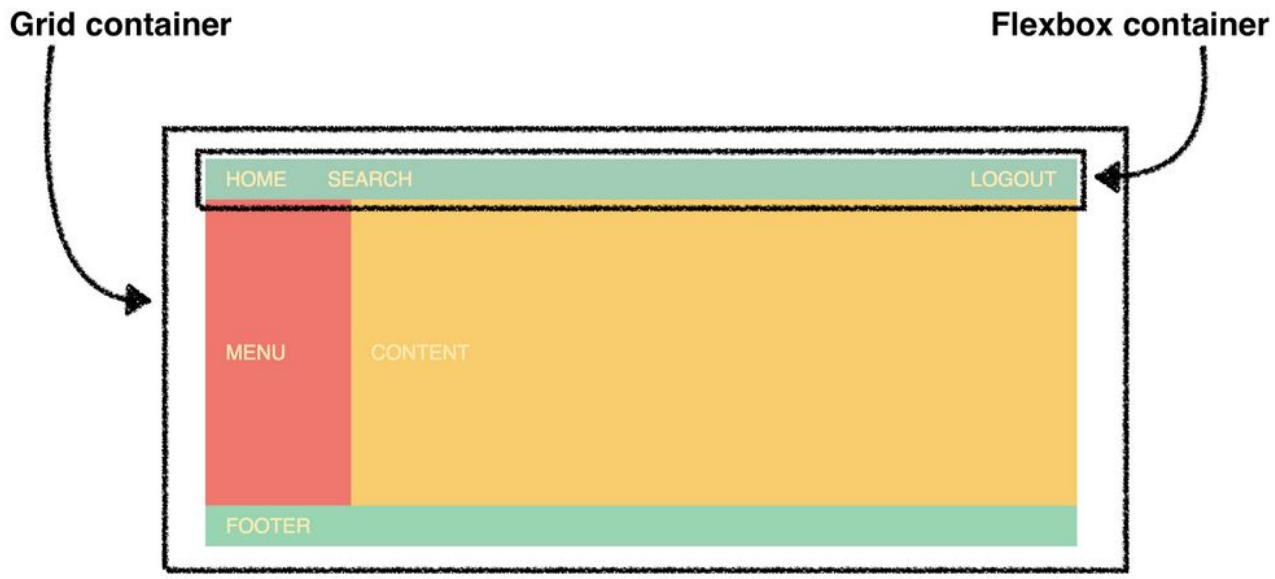
```
.grid-container {
  display: grid;
  grid-gap: 50px 100px;
}
```

> **Grid areas:**




```
.item-a {
  grid-area: header;
}
.item-b {
  grid-area: main;
}
.item-c {
  grid-area: sidebar;
}
.item-d {
  grid-area: footer;
}

.container {
  grid-template-columns: 50px 50px 50px 50px;
  grid-template-rows: auto;
  grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer";
}
```

❖ **Flex + Grid:**


Grid container
Flexbox container
HOME
SEARCH
LOGOUT
MENU
CONTENT
FOOTER

❖ Media

```
<img src="dog.jpg" alt="doggie">
```

> **CSS backgrounds:**

```
background-image: url(../images/clouds.jpg);
```

> **HTML vs CSS:**

Images in HTML

- Images that are part of the content belong in the HTML
- Use the HTML: `<img ...>` element

Images in CSS

- Images used for style, decoration (eye-candy) or otherwise "not content" belong in the CSS as background images
- Use the CSS property: `background-image: url("...");`

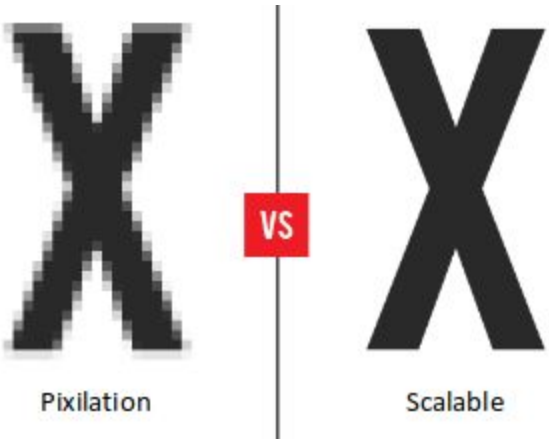
> **Raster vs Vectors:**

Raster (bitmaps)

- JPGs, GIFs, PNGs
- Pixels
- Detailed images
- Photographs
- Realistic
- Can be large file sizes
- Limited scalability
- Not inherently mobile-friendly

Vector (line drawings)

- SVGs
- Lines, curves, fills
- Line art
- Logos, drawings
- Simplistic
- Small file sizes
- Scalable
- Mobile-friendly



> **File Types:**

GIF (.gif)

- Graphics Interchange Format
- Best used for line art and logos
- Maximum of 256 colors
- Can use a transparent color (layer)
- Uses lossless compression
- Can be animated

JPEG (.jpg)

- Joint Photographic Experts Group
- Best used for photographs
- Up to 16.7 million colors
- Uses lossy compression
- Cannot be made transparent
- Cannot be animated

PNG (.png)

- Portable Network Graphic
- Can be used for both photographs or line art and logos
- Up to 16.7 million colors
- Use lossless compression
- Can use a transparent color (layer)
- Can be animated (but not very well)
- Combines the best of GIF & JPEG
  - Doesn't compress as well as JPG (large file sizes!)

❖ **Basics of PHP:**

>Server-side Includes (SSI): “include” html content for common header, sidebar, footer, etc. The purpose is to reduce the amount of code you duplicate in webpages.

>Trigger: the server looks for scripts on .php files with include command:

```
<?php include "inc/more-content.inc"; ?>
```

>Industry Standards: file extension: .inc inside inc/ folder.

>Configure a web server: LAMP, MAMP and WAMP are examples of “software stack” with Apache, MySql, PHP, etc that you can download to turn your laptop into a private web server.

❖ HTML Forms

> 1. Html Form [requires action and get: {get, post}]

```
<form action="/action_page.php" method="get">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
</form>
```

-method = "get/post": get is default value, form data is passed in URL. **Post** is more secure, form data passed in HTTP Entity body.
-action= "....php": specifies the server-side program or script that will process your form data.

-Input Types {button, checkbox, color, date, datetime-local, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week} **html5 does type checking so that the form doesn't get submitted if you didn't enter an email, for example.**

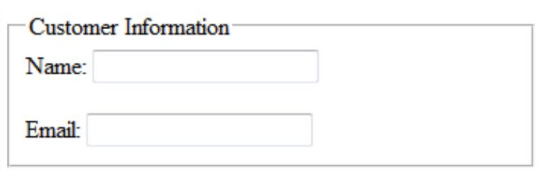
```
<textarea rows="4" cols="50">
```

-name= "email" : give the data a name so it can be retrieved with the variable name later. Typically use the SAME name for id= "name" [best practice, always use an id for JS].

>Fieldset and Labels

-Labels [the for attribute targets the id of the input]

```
<label for="email">Email: </label>

<input type="text" name="CustEmail" id="email">
```



- The Fieldset Element
  - Container tag
  - Creates a visual group of form elements on a web page
- The Legend Element
  - Container tag
  - Creates a text label within the fieldset

use <legend> to label the <fieldset>

```
<fieldset>

  <legend>Customer Information</legend>

  <label>Name: </label>
  <input type="text" name="name" id="name">

 <label>Email: </label>
  <input type="text" name="email" id="email">

</fieldset>
```

> 2. PHP- Server Side processing of HTML Forms

> PHP:

- a server-side scripting language
- PHP must be installed and running on the web server for it to work
- the extension .php is used instead of .html. It tells the web server that there maybe PHP code in the file. The web server looks at .php files and runs the PHP code if there is php code.

```
<?php //PHP code here ?>
```

or

```
<?php
//PHP code here, on
//  multiple lines
?>
```

> Variables in PHP: $var = value;

```
$myVariable = "Hello";
```

> Concatenation: $var .= " Concat";

```
$myVariable = "Hello";
$myVariable .= " Professor!";
```

>PHP Commands:

- mail() : [TO, SUBJECT, BODY FROM]-send mail using the server's mail server software

```
mail("someone@example.com",
  "The subject line",
  "Hello World!",
  "From:<someoneElse@example.com>
);
```

- echo : inject html on the webpage

```
echo "these words will appear on the webpage"
echo $myWords
```

> **PHP Superglobals: always accessible, regardless of scope.** [these are method types on the HTML form] **Use them to pull data from HTML forms.**

$_POST[] or **$_GET[]**

File 1: index.html

```
<form method="post" action="welcome.php">
   Name: <input type="text" name="name">
   E-mail: <input type="text" name="email">
   <input type="submit">
</form>
```

File 2: welcome.php

```
Welcome <?php echo $_POST["name"]; ?>
Your email address is: <?php echo $_POST["email"]; ?>
```

> **Scrub incoming data:**

Users can trick servers to run code by entering it into a form and submitting it. Scrub the input to protect the web server!

- Use the PHP trim() and stripslashes() commands
  - E.g.

```
$message = Trim(stripslashes($_POST['message']));
```

❖ **JavaScript**

> **Use:** Javascript is an **Object-oriented programming** language commonly used to create interactive effects within web browsers. It deals with customizing the **behavior** of a website, focusing on **interaction** and **usability.**
Note: Javascript != Java

> **History:** Netscape collaborated with Sun Microsystems in the 90's to make a web programming language complementary to Java. It was first called Mocha, then LiveScript and finally Javascript.

> **DHTML (Dynamic HTML):** web pages with javaScript are called DHTML.

> **JavaScript has both front-end and back-end usability.**
> **It is interpreted by the client software.**

> **Three techniques to implement JavaScript:**

- Link JavaScript code from an EXTERNAL file

```
<script src="js/scripts.js"></script>
```

- INTERNAL : Place JavaScript code between script tags

```
<script>
  alert("Hello World!");
</script>
```

- Place JavaScript code as part of an EVENT attached to an HTML element (i.e. click)

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

> **HTML DOM Events:** these allow JavaScript to **register different event handlers on elements** in an HTML document. You use these events to “fire”/ “trigger” javascript functions.

More events here: https://www.w3schools.com/jsref/dom_obj_event.asp

Events

```
<button onclick="myFunction();">
   Click here
</button>
```

Mouse

onclick, oncontextmenu, ondblclick, onmousedown, onmouseenter, onmouseleave, onmousemove, onmouseover, onmouseout, onmouseup

Keyboard

onkeydown, onkeypress, onkeyup

Frame

onabort, onbeforeunload, onerror, onhashchange, onload, onpageshow, onpagehide, onresize, onscroll, onunload

Form

onblur, onchange, onfocus, onfocusin, onfocusout, oninput, oninvalid, onreset, onsearch, onselect, onsubmit

Drag

ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop

Clipboard

oncopy, oncut, onpaste

Media

onabort, oncanplay, oncanplaythrough, ondurationchange, onended, onerror, onloadeddata, onloadedmetadata, onloadstart, onpause, onplay, onplaying, onprogress, onratechange, onseeked, onseeking, onstalled, onsuspend, ontimeupdate, onvolumechange, onwaiting

Animation

animationend, animationiteration, animationstart

Miscellaneous

transitionend, onmessage, onmousewheel, ononline, onoffline, onpopstate, onshow, onstorage, ontoggle, onwheel, ontouchcancel, ontouchend, ontouchmove, ontouchstart

> JavaScript Popup Boxes, Functions and Variables:

- An **alert** is used to give information. User must click "OK" to proceed.
- A **confirm** is used to verify something.
  User will have to click either "OK" or "Cancel" to proceed
  ("OK" returns true, "Cancel" returns false)
- A **prompt** is used to gather information from the user.
  User will have to click either "OK" or "Cancel" to proceed
  ("OK" returns the input value, "Cancel" returns null)

Functions

```
function addNumbers(a, b) {
    return a + b; ;
}
x = addNumbers(1, 2);
```

Edit DOM element

```
document.getElementById("elementID").innerHTML = "Hello World!";
```

Output

```
console.log(a);              // write to the browser console
document.write(a);           // write to the HTML
alert(a);                    // output in an alert box
confirm("Really?");          // yes/no dialog, returns true/false depending
prompt("Your age?","0");     // input dialog. Second argument is the initia
```

Comments

```
/* Multi line
   comment */
// One line
```

Variables

```
var a;                          // variable
var b = "init";                 // string
var c = "Hi" + " " + "Joe";     // = "Hi Joe"
var d = 1 + 2 + "3";            // = "33"
var e = [2,3,5,8];              // array
var f = false;                  // boolean
var g = /()/;                   // RegEx
var h = function(){};           // function object
const PI = 3.14;                // constant
var a = 1, b = 2, c = a + b;    // one line
let z = 'zzz';                  // block scope local va
```