

# HTML & CSS

(via nemldog.com)

(HTMLdog's beginner HTML tutorial)

## Tags, Attributes and Elements

`<!DOCTYPE html>`

Document type declaration - lets browser know which flavor of HTML you're using. IMPORTANT. if not present, browsers assume you know what's up

`<html>`

Opening tags: Everything enclosed is a HTML document

`<body>`

This is my first web page

`</body>`

Main content that will appear in the browser window is within these tags

`</html>`

Note:

Some tags like `<br>` is self closing. `<br />` is a remnant of XHTML so don't confuse yourself.

Attributes: Extra bits of info for tags.

e.g.: `<tag attribute="value">Margarine</tag>`

↳ Not essential but good to have.

Elements: Bits that make up the web page. Everything between (& inclusive of) `<body>` and `</body>` tags is the body element. e.g.: `<title>Purple</title>` is a title element

## Title

`<head>` tag appears before `<body>` tag. `<head>` contains info about the page.

`<title>` element will appear on a tab or title bar of the window.

## Paragraphs

- The browser doesn't distinguish text on diff lines. That is,   
there → "Hi there" in the browser
- Use `<p>` tag for paragraphs!

`<em>` for emphasis, and `<strong>` for strong importance.

italics

bold in general

- `<br>` for line breaks. Note: Shouldn't be used if two blocks of text are intended to be separate from one another (consider `<p>` tag instead)

## Headings

- If you have documents with genuine headings, then consider using: `h1`, `h2`, `h3`, `h4`, `h5`
- `h1` is the almighty emperor of headings
- `h6` is the lowest pleb.

Note:

`h1` tag is only used once!

`h2` to `h6` can be used as often, but should always be used in order, as intended.

e.g. `h4` is a sub-heading of `h3`, which is a sub-heading of `h2`

## Lists

Three types of list:

- ↳ Unordered lists → non-sequential (usually bullets)
- ↳ Ordered lists → sequential (normally incremental numbers)
- ↳ definition lists

Lists can also be included in Lists!

```
<ul>
  <li> To learn HTML </li>
  <li> To show off
    <ol>
      <li> To my boss </li>
      <li> To my friends </li>
    </ol>
  </li>
  <li> Witness me greatness </li>
</ul>
```

## Links

HTML → "hypertext" - a system of linked text

The anchor tag `<a>` is used to define a link, but you also need to add sth to the anchor tag - the destination of the link.

e.g. `<p><a href="http://www.htmldog.com">HTML Dog</a></p>`  
(like above)

Link (href) can be → absolute  
→ relative to current page. e.g. `<a href="flyingmoss.html"></a>`

It can link to anywhere on web

**Note:** Can link to another part of the same page by adding **id** attribute to any tag.  
e.g. `<h2 id="moss">Moss</h2>`, link using `<a href="#moss">go to moss</a>`

## Image

: use **img** tag!

``

Tells us where to find the image similar to a tag; can be absolute or relative

width="120" height="40"  
Necessary bcs if excluded, browser calculate size as the img loads, instead of when page loads.  
layout of document may jump arnd!

Alternative description.  
Accessibility consideration.  
Providing info for PPI.  
e.g. visually impaired.

**Note:** self-closing!

### JPEG

- Uses math algo to compress & distort image slightly. Lower compression, higher file size, but clearer!
- Typically used for images such as photo graphs

### GIF

- Can have ≤ 256 colors, but they maintain colors of original image. ↓ no. of colors, ↓ file size.
- Allows any pixel to be transparent
- Used for images with solid colors, such as icons or logos

### PNG

- Replicates colors, much like a GIF, altho it allows 10<sup>16</sup> colors & alpha transparency (area could be 50% transparent)
- Typically used for versatile images in more complex designs BUT not fully supported by some older browsers

Try to strike balance betw img quality and size!



# Tables

Example code:

- `<table>` element defines the table.
- `<tr>` element defines a table row
- `<td>` element defines a data cell.
- and it must be enclosed in `<tr>` tags!

e.g.

```
<table>
<tr>
  <td> Row 1, Cell 1 </td>
  <td> Row 1, Cell 2 </td>
</tr>
<tr>
  <td> Row 2, Cell 1 </td>
  <td> Row 2, Cell 2 </td>
</tr>
</table>
```

# Forms

Used to collect data inputted by a user. Can be used as an interface for a web application (Tend to be used in conjunction with a programming language to process info)

Basic methods:

Form

↳ Defines the form, and if you are submitting, an **action** attribute is needed to tell the form where its contents will be sent to.

↳ **Method** attribute tells how the data in it is going to be sent. Possible values:

Get:

- ↳ default
- ↳ latches form info onto a web address
- ↳ Used for shorter chunks of non-sensitive information

Post:

- ↳ invisibly sends the form's information
- ↳ used for lengthy, more secure submissions

eg:

```
<form action="Script.php" method="post">
```

```
</form>
```

input: daddy of the form world lol

Text

- ↳ Simply `<input>` works
- ↳ **value** attribute to set initial text in text box

Password

- ↳ Similar to text box, but hidden characters + yep

Checkbox

- ↳ Can be toggled.
- ↳ Can have **checked** attribute: Makes initial state switched on

Radio

- ↳ Similar to checkbox
- ↳ Can have **checked** attribute

Submit

- ↳ when selected, will submit the form.
- ↳ control text that appears with **value** attribute
- ↳ `<input type="submit" value="Is a button">`

→ self closing!

Basic Inputs: `<input type="x">`

## textarea

Large, multi-line textbox. Anticipated number of rows can be defined with **rows** and **cols** attribute. Can use CSS here!

e.g.

```
<textarea rows="5" cols="20">
  A big load of text
</textarea>
```

Forms the initial value!

## Select

**Select** tag works with **option** tag to make drop-down select boxes

An **option** tag can also have a **selected** attribute, to start off with one of the items already selected (doesn't work in Firefox tho)

If form submitted:  
the value is either:

↳ enclosed in the tag

↳ specified by **value** attribute

e.g.

```
<select>
  <option> Opt 1 </option>
  <option value="second option">
    Opt 2
  </option>
</select>
```

## name

Without the **name** attribute, the form-handling script will ignore all of the tags.

e.g.:

```
<input type="text" name="talking sponge">
```

## Wrapping up

```
<!-- This is a comment -->
```

# HTML Doc CSS Beginner tutorial

CSS, or Cascading Style Sheets, is a way to present HTML.

HTML is meaning or content, CSS is presentation

Has a format of: 'property: value'

## Applying CSS

(bad) inline: `<P style="color: red"> text </P>`

But not good! Avoid inline!! HTML should be stand-alone, presentation-free

(bad) internal: In HTML file, inside the `head` element, `style` tags surround all of the same styles.

```
<head>
<style>
  P {
    color: red;
  }
</style>
</head>
```

Also try to avoid!!

external: File saved as "... .css", Linked to HTML  
 eg. file named "style.css", can be linked like this:

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

## Selectors, Properties, Values

: HTML has tags, CSS has selectors

HTML selectors: names of HTML tags; used to change the style of a specific type of element

eg. `body { font-size: 14px; color: navy; }`

Selector: `body`  
 Properties: `font-size`, `color`  
 Values: `14px`, `navy`

Note: value given to property following a colon (NOT equals). Semi-colons to separate properties

Common Property-specific units:

- ↳ `px`: doesn't mean computer pixels. It specifies size for non-zoomed browser, but can still auto-increase or auto-decrease all the time
- ↳ `em`: `font-size: 2em`; means two times current font size
- ↳ `pt`: points, typically in printed media
- ↳ `%`



# Colors

CSS brings 16,777,216 colors. It can take 3 forms: ① name, ② RGB value, or ③ a hex code.

Following 4 values ~~also~~ produce same result:

- red
- rgb(255, 0, 0)
- rgb(100%, 0%, 0%)
- #ff0000
- #f00

transparent is also a valid value for predefined color names

Hex number prefixed with hash character (#):

↳ Can be 3 or 6 digits.

↳ 3-digit version is easier to decipher (corresponds to RGB)

## Color and background-color

American english! NOT 'colour'

e.g. body {  
font-size: 12px;  
color: navy;  
h1 {  
color: #fec;  
background-color: #009;

## Text

They are all properties

### Font-family

Users browser has to be able to find the font you specify. Has to be on THEIR computer.

Can specify using commas: font-family: arial, helvetica, "times new roman";  
Looks through arial first, then helvetica. For font with spacing, use quotes.

### Font-size

Sets the font size. Be careful! It makes more sense to use headings

### Font-weight

Bold or not. Some values: bold, bolder, 100, 400 (same as normal), 700 (same as bold). Advice: just stick to bold or normal!

### Font-style

Italics or not.

### Text-decoration

Does text have line running around/in it? Values: underline, overline, line-through

### Text-transform

Changes the case. Values: capitalize → first letter or word becomes uppercase, uppercase, lowercase

Spacing for text:

### Letter-spacing / word-spacing

Spacing between. Value can be length or normal.

### Line-height

Sets height in an element (for lines) without adjusting size of font. Values: number (multiple of font size), length, percentage, or normal.

### Text-align

Align text inside element to left, right, center or justify.

### Text-indent

Indent the first line of a paragraph, to a given length or %

## Margins and Padding

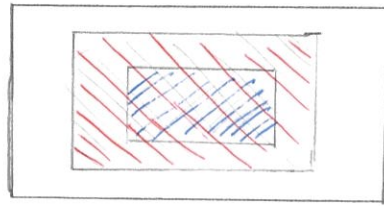
Margin: Space **Outside** something



Padding: Space **inside** something

consider:

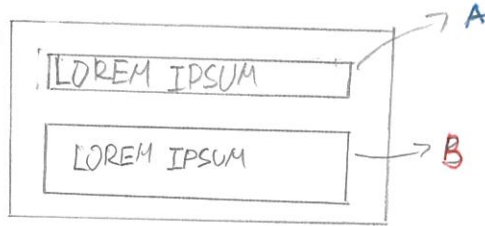
```
h2 {
  background-color: #ccc;
  margin: x px;
  padding: y px;
}
```


⇒



 should have a margin that is larger than  (bcs space **outside** something is larger, so everything gets 'pushed', I think).

This code leaves  $x$ -pixel width space around secondary header, and header itself is far from the ~~the~~  $y$ -pixel width padding

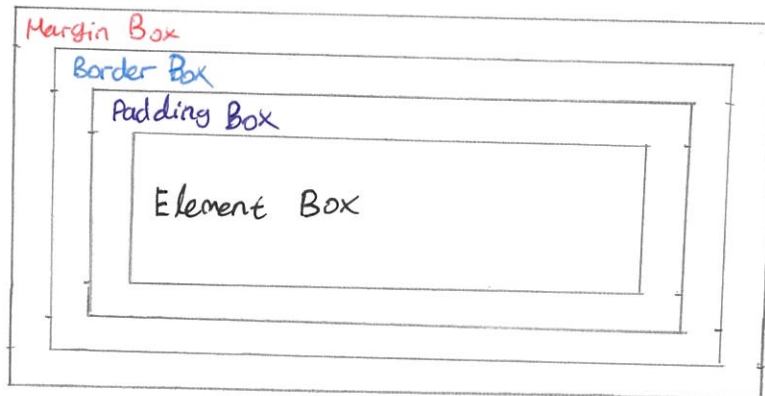



\* Content A  has a smaller padding than content B. In fact, content A may actually have a padding of 0px. (No space inside something). Notice how in B, the left-indented text gets pushed by approx the same amt.

Four sides of an element can be set individually by using:

- margin-top, margin-right, margin-bottom, margin-left
- padding-top, padding-right, padding-bottom, padding-left

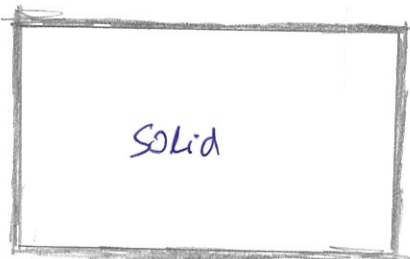
## The Box Model:



Middle: content area (e.g.  image) surrounding it is padding, which is surrounded by border, and finally surrounded by margin.

## Borders

Use **border-style** property. Values can be solid, dotted, dashed, double, groove, ridge, inset, outset.



`border-width` to set width of border.

See also: `border-right-width`, `border-left-width`, `border-top-width`, `border-bottom-width`

`border-color` to set color

e.g.

`h2 {`

`border-style: dashed;`

`border-width: 3px;`

`border-left-width: 10px;`

`border-right-width: 10px;`

`border-color: red; red;`

`}`

} →

These overrides

`border-width`!

If these are put above,  
reverse happens (`border-width`  
overrides)

### Wrapping up

`<!-- Insert notes here -->`



# HTML Dog CSS Intermediate Tutorial

contents: Class & ID selectors, Grouping & Nesting, Pseudoclasses, Shorthand properties, Background images, Specificity, Display, Pseudo elements, Page layout

## Class and ID selectors

In the beginner tut, we only looked at HTML selectors (representing tags)

We can define selectors in the form **class** and **ID** selectors.

Benefit: Select **same HTML element**, but present it differently depending on **class** or **ID**

CSS may look like this:

```
#top {
  background-color: #ccc;
  padding: 20px;
}

.intro {
  color: red;
  font-weight: bold;
}
```

for

HTML:

```
<div id="top">
  <h1>Chocolate curry</h1>
  <p class="intro"> This is my recipe </p>
  <p class="intro"> Mmm looks good </p>
</div>
```

Preceded by Full Stop (".")

Preceded by hash character ("#")

can identify more than one element

to identify one element

Can apply a selector to specific HTML element: state HTML selector first!

e.g.:

```
p.intro {
  // some
```

Applied to Paragraph elements that have the class "intro"

## Grouping and Nesting

: To simplify your code!

**Grouping**: Same properties to selectors without having to repeat

```
h2 {
  color: red;
}

.thisClass {
  color: red;
}

.thatClass {
  color: red;
}
```

equivalent  
⇒

apply commas!

```
h2, .thisClass, .thatClass {
  color: red;
}
```

# Abstraction

is a hashtag here  
not an ID selector

**Nesting**: To specify properties to selectors within other selectors

e.g.

```
#top {
  background-color: #ccc;
  padding: 1em;
}

#top h1 {
  color: #ff0;
}

#top p {
  color: red;
  font-weight: bold;
}
```

By separating selectors with spaces, we are saying:

"h1 inside ID top is colour #ff0"

"p inside ID top is red and bold"

This is suitable for code ② of the Chocolate Curry in Page 9 (prev page), but you can remove `class="intro"`

(can get complicated so may take practice)

## Pseudo classes

Bolted on to selectors to specify a state or relation to selectors. [I think these are native to CSS i.e. not self-defined]

Form:

```
Selector: pseudo_class {
  Property: value;
}
```

⇒ Simply add a colon in between Selector and pseudo class

## Links

```
a: link {
  color: blue;
}

a: visited {
  color: purple;
}
```

targets unvisited links

targets visited links

⇒ This CSS apply colors to all link in a page depending on whether user has visited that page or not.

## Dynamic Pseudo Classes

Also commonly used for links. To apply styles when something happens to something.

**active**: Something activated by user, such as when a link is clicked on.

**hover**: Something passed over by an input from user, such as when cursor moves over a link

**focus**: Something gains focus, that is when it is selected by, or is ready for keyboard input

→ Most often used on form elements but can be used for links. Consider those users not using mouse, eg. keyboard only

## First Children

Targets something only if it is the very first descendant of an element.

e.g.

```
<body>
  <p> I'm the first child of- body </p>
  <p> (inferior) second child </p>
  ...
</body>
```

Targets this only

```
p: first-child {
  font-weight: bold;
  font-size: 40px;
}
```

Note: CSS3 delivered a whole new set of Pseudo classes:

last-child, target, first-of-type

## Shorthand Properties

Some properties allow a string of values, separated by spaces

### Margins and Padding

margin amalgamates margin-top, margin-right, margin-bottom and margin-left

Form: Property: top right bottom left; ★ the order is impt!

Mnemonic: The Red Brother Lied

sad :(

clockwise if you think abt it actually,

```

P {
  margin-top: 1px;
  margin-right: 5px;
  margin-bottom: 10px;
  margin-left: 20px;
}
  
```

⇒

```

P {
  margin: 1px 5px 10px 20px;
}
  
```

Padding can be used in exactly the same way.

Also, you can state two values ⇒ padding: 1em 10em  
top & bottom right & left

Mnemonic: T for Top & Bottom, R for Right & Left  
The Deal

### Borders

border-width can be used the same way as padding and margin.  
Another way:

```

P {
  border-width: 1px;
  border-color: red;
  border-style: solid;
}
  
```

⇒

```

P {
  border: 1px red solid;
}
  
```

Mnemonic: World Champion - Shio

Note: w/c/s combi can also be applied to border-top, border-right

### Font

```

P {
  font-style: italic;
  font-weight: bold;
  font-size: 12px;
  font-family: courier;
}
  
```

font-style    font-weight    font-size    font-family

combines font-style, font-weight, font-size line-height font-family

For some reason it is divided by slash...

Mnemonic: Statistics work shop hates failing  
Doesn't make sense but num

## Background Images: it's different from img HTML element!

Shorthand property for background:

```

body {
  background: white url(http://www.html5dog.com/images/bg.gif) no-repeat top right;
}
  
```

color    image    repeat    position

background-color: self-explanatory

background-image: Location of image

background-repeat: How the image repeats itself:

↳ repeat: "tile" across whole background

↳ repeat-y: repeat on y-axis

↳ repeat-x: repeat on x-axis

↳ no-repeat: single instance

background-position: top, center, bottom, left, right, a length, percentage or any sensible combo such as top right.

Can specify things like attachment, clip, origin and size.

Note: Background images can be used for most HTML elements. Can use effectively. e.g. visit HTML5 dog and see the magnifying glass in the search box, and as icons in the top left corner



## Specificity

When there is conflict for application of CSS rules, browser follows some rules to determine which one is most specific

If the selectors are the same then the last one takes precedence

`P { color: red; }`  
`P { color: blue; }` → Takes precedence, ∴ elements colored blue.

But, this usually won't happen. Consider nested selectors

`div P { color: red; }`  
`P { color: blue; }` → Is last, but does not take precedence!

↳ Text will actually be **red**! The first selector has a higher specificity.

The more specific a selector, the more preference it will be given

## Calculating specificity

ID selectors: 100  
 Class selectors: 10  
 HTML Selector: 1

Values, add them up to get total specificity. The higher, the better

⇒ implies ID is v. specific, followed by class then HTML

e.g.

`body P` = 1

↳ `div p, tree` =  $1 + 1 + 10 = 12$

`body div P` =  $1 + 1 + 10 = 12$

↳ `#baobab` = 100

↳ `.tree` = 10

↳ `body #content .alternative P` =  $1 + 100 + 10 + 1 = 112$

## Display

9 May 2019

Browsers default visual representation of most HTML elements consist of varying font styles, margins, padding and essentially, display types

Types of display: inline, block, none.

**inline**: Boxes that are displayed follow the flow of a line.

**block**: Makes a box stand alone, fitting the entire width of box with line break.

↳ Allows manipulation of height, margins and padding.

Note: inline-block will make boxes inline but still allow formatting flexibility.

**none**: Doesn't display boxes at all! Useful to "turn off" display, e.g. alternating.

↳ `display: none` ⇒ Element's box is completely out of play.

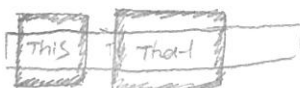
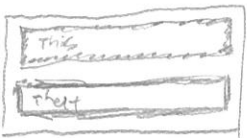
Visibility: `hidden` ⇒ keeps the box and its flow in place, without representing content.

There are tables, list-item and run-in but this seems less important.

Block:

Inline:

Inline-block:



## Pseudo elements

They are like pseudo classes. It has this form:

Selector: PseudoElement {

Property: Value;

}

### First-letter & first-line

The **first-letter** Pseudo element is applied to first letter inside a box,  
**first-line** is applied to the top-most displayed line in a box (It doesn't require line breaks)

e.g.:

```
p {
  font-size: 12px;
}
```

```
p: first-letter {
  font-size: 24px;
  float: left;
}
```

```
p: first-line {
  text-decoration: underline;
}
```



Over the years,  
I have not found  
anyone else worthy  
of wielding Mjolnir,  
except perhaps one person.

Note: CSS3 specs suggest  
Pseudo elements should include  
two colors to differentiate  
with Pseudo elements.  
But it seems like Safari  
not backwards compatible

### Before and after Content

**before** and **after** are used in conjunction with the **content** property to place content without touching the HTML. Use sparingly! You are using content solely for presentation.

The value of **content** can be:

- ↳ Open-quote
- ↳ String enclosed in quotation marks
- ↳ Close-quote
- ↳ Any image, using `url(imageName)`

```
blockquote: before {
  content: open-quote;
}
```

```
blockquote: after {
  content: close-quote;
}
```

Essentially, a new box to play with!

```
li: before {
  content: "Pow!";
  background: red;
  color: #f00;
}
```

```
p: before {
  content: url(images/jam.jpg);
}
```

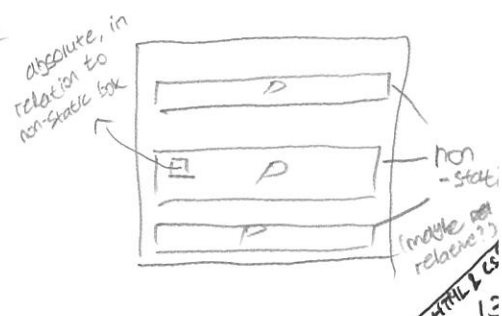
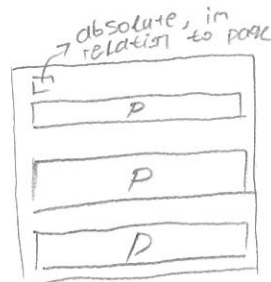
**Page layout**: Essentially, taking a chunk of your page & shove it wherever you choose

**Positioning**: **position** property is used to define whether a box is absolute, relative

- ↳ **static**: default value and renders a box in the normal order
- ↳ **relative**: like static, but box can be offset from position with properties: **top, right, bottom, left**
- ↳ **absolute**: pulls a box <sup>out</sup> into its own crazy little world. **top, right, bottom, left** are also used.
- ↳ **fixed**: like absolute, but will reference to browser window. Fixed boxes stay exactly where they are on screen, even after scrolling.

Note: Absolutely positioned box is still relative from edge of the page. Page doesn't have to be container. A box will be absolutely positioned in relation to any non-static positioned box!

Downside of these boxes race page.





Downside: They live in a world of their own; can't determine accurately!

When using relative values for widths and sizes, we need to abandon all hope of placing anything! (e.g. footer).

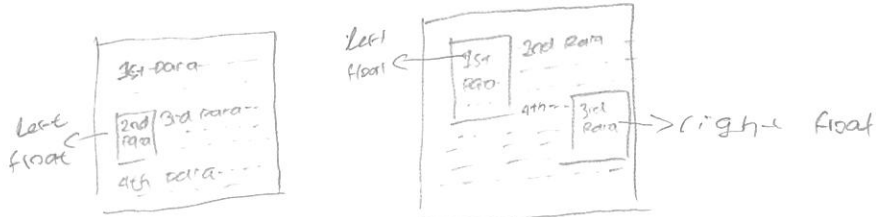
Enter float!

## Floating

Floating a box will shift it to the right or left of a line, with surrounding content flowing around it

• Normally for smaller chunks like a line, but bigger chunks like columns is good too.

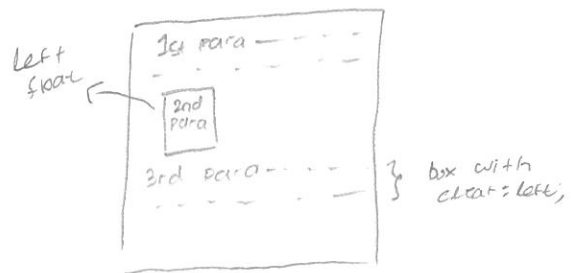
float: left;  
float: right;



But, if you don't want the next box to wrap around the floating

objects: apply clear

Clear: left; - clear left-floated box  
Clear: right; - clear right-floated box  
Clear: both; - clear both floated boxes



e.g.

```
<div id="footer">
  <p> This is a footer </p>
</div>
```

```
#footer {
  clear: both;
}
```

Creates footer that will appear underneath all columns, regardless of the length of any of them

While this tut emphasises larger "chunks", these methods can be applied to any box within boxes too!