



# CS472 Web Programming

## Lecture 3: Layout



---

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

---

## Maharishi University of Management -Fairfield, Iowa © 2016



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.

# Wholeness Statement

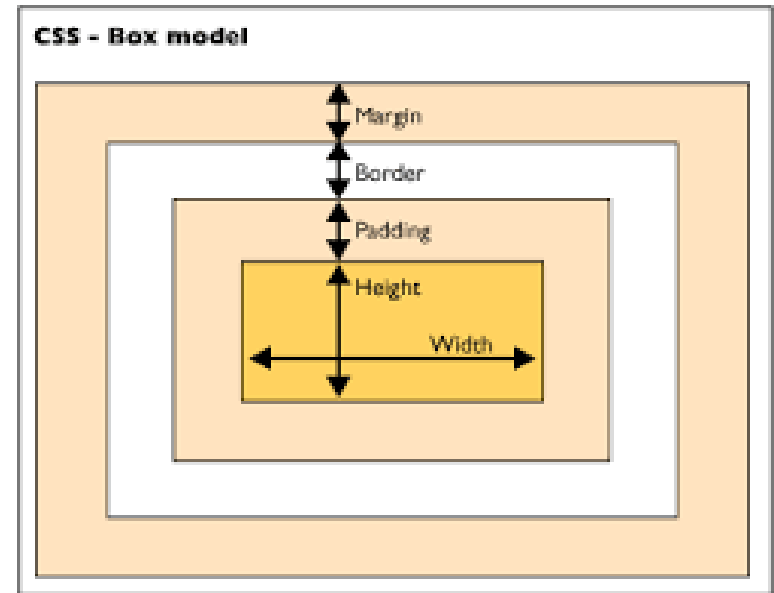
---

In this lecture we will discuss the different tools CSS provides for creating a layout. There are a variety of ways to position an element; most of them are based on taking a block level element and placing it in relation to some other block.

*Through familiarity with the language of nature, the individual gains complete support of nature.*

# The CSS Box Model

- ▶ For layout purposes, every element is composed of:
  - The actual element's **content**
  - A **border** around the element
  - **padding** between the content and the border (inside)
  - A **margin** between the border and other content (outside)
- ▶  $\text{width} = \text{content width} + \text{L/R padding} + \text{L/R border} + \text{L/R margin}$
- ▶  $\text{height} = \text{content height} + \text{T/B padding} + \text{T/B border} + \text{T/B margin}$
- ▶ The standard `width` and `height` properties refer **ONLY** to the content's width and height.





# CSS properties for borders

---

```
h2 { border: 5px solid red; }
```

This is the best WAP course!

Property	Description
border	thickness/style/size of border on all 4 sides

- ▶ **thickness** (specified in px, pt, em, or thin, medium, thick )
- ▶ **style** (none, hidden, dotted , dashed , double , groove , inset , outset , ridge , solid )
- ▶ **color** (specified as seen previously for text and background colors)

# More border properties

---

Property	Description
border-color, border-width, border-style	specific properties of border on all 4 sides
border-bottom, border-left, border-right, border-top	all properties of border on a particular side
border-bottom-color, border-bottom-style, border-bottom-width, border-left-color, border-left-style, border-left-width, border-right-color, border-right-style, border-right-width, border-top-color, border-top-style, border-top-width	properties of border on a particular side
<a href="#">Complete list of border properties</a>	

# Border example 2

---

```
h2 {  
  border-left: thick dotted #CC0088;  
  border-bottom-color: rgb(0, 128, 128);  
  border-bottom-style: double;  
}
```

**This is the best WAP course!**

- ▶ each side's border properties can be set individually
- ▶ if you omit some properties, they receive default values (e.g. border-bottom-width above)



# Dimensions

---

- ▶ For **Block elements and `img` element only**, set how wide or tall this element, or set the max/min size of this element in given dimension.

width, height, max-width, max-height, min-width, min-height

```
p { width: 350px; background-color: yellow; } h2 {  
width: 50%; background-color: aqua; }
```

This paragraph uses the first style above.

An h2 heading

Using **max-width** instead of **width** in this situation will improve the browser's handling of small windows. This is important when making a site usable on mobile. [https://www.w3schools.com/css/tryit.asp?filename=trycss\\_max-width](https://www.w3schools.com/css/tryit.asp?filename=trycss_max-width)

# box-sizing



- ▶ `box-sizing: content-box;` - initial and default value. The width and height properties are measured including only the content, but not the padding, border or margin.
- ▶ `box-sizing: border-box;` The width and height properties include the content, the padding and border, but not the margin. Note that padding and border will be inside of the box
- ▶ Q: Why is Hooray indented?
- ▶ [responsive design advantage demo](#)

```
.div3 {  
  width: 300px;  
  height: 100px;  
  border: 1px solid blue;  
  box-sizing: border-box;  
}  
  
.div4 {  
  width: 300px;  
  height: 100px;  
  padding: 50px;  
  border: 1px solid red;  
  box-sizing: border-box;  
}
```

Both divs are the same size now!

Hooray!

# Rounded corners `border-radius`



```
p {  
  border: 3px solid blue;  
  border-radius: 12px;  
}
```



- ▶ Each side's border radius can be set individually, separated by spaces
  - ▶ **Four values:** top-left, top-right, bottom-right, bottom-left
  - ▶ **Three values:** top-left, top-right and bottom-left, bottom-right
  - ▶ **Two values:** top-left and bottom-right, top-right and bottom-left
  - ▶ **One value:** all four corners are rounded equally

# Padding

---

- ▶ The padding shorthand property sets all the padding properties in one declaration. Padding shares the background color of the element. This property can have from one to four values:

```
padding:10px 5px 15px 20px; /* Top, right, bottom, left */
padding:10px 5px 15px; /* Top, right and left, bottom */
padding:10px 5px; /* Top and bottom, right and left */
padding:10px; /* All four paddings are 10px */
```

- ▶ padding-bottom, padding-left, padding-right, padding-top

```
h1 { padding: 20px; }
h2 {
    padding-left: 200px;
    padding-top: 30px;
}
```

# Margin

---

- ▶ Margins are always transparent. This property can have from one to four values:

```
margin:10px 5px 15px 20px; /* Top, right, bottom, left */  
margin:10px 5px 15px; /* Top, right and left, bottom */  
margin:10px 5px; /* Top and bottom, right and left */  
margin:10px; /* All four margins are 10px */
```

- ▶ margin-bottom, margin-left, margin-right, margin-top

```
h1 {margin: 20px; }  
h2 {  
    margin-left:    200px;  
    margin-top:     30px;  
}
```

- ▶ See example: [lesson3\\_examples/paddingmargin.html](lesson3_examples/paddingmargin.html)
  - ▶ Inspect element in console



# Margin Collapse

---

- ▶ Vertical margins on different elements that touch each other (thus have no content, padding, or borders separating them) will collapse, forming a single margin that is equal to the greater of the adjoining margins.
  1. Collapsing Margins Between Adjacent Elements
  2. Collapsing Margins Between Parent and Child Elements

```
h1 { margin: 0 0 25px 0; }
```

```
p { margin: 20px 0 0 0; }
```

```
h1 { margin: 0 0 25px 0; }
```

```
p { margin: -20px 0 0 0; }
```

See example:

[lesson3\\_examples/margincollapse.html](#)

[lesson3\\_examples/margincollapse2.html](#)

[lesson3\\_examples/margincollapse3.html](#)

# Centering a block element: auto margins



```
<p> Lorem ipsum dolor sit amet, consectetur  
    adipisicing elit, sed do eiusmod tempor incididunt  
    ut la-bore et dolore magna aliqua.
```

```
</p>
```

```
p {  
  border: 2px solid black;  
  margin-left: auto;  
  margin-right: auto;  
  width: 33%;  
}
```

- ▶ works best if width is set (otherwise, may occupy entire width of page)
- ▶ to center inline elements within a block element, use text-align: center;

## Details about **block** boxes

---

- ▶ By default **block** elements take the entire width space of the page unless we specify.
- ▶ To align a **block** element at the **center** of a horizontal space you must set a **width** first, and **margin: auto;**
- ▶ **text-align** does not align **block** elements within the page.



# Details about **inline** boxes

---

- ▶ size properties (width, height, min-width, etc.) are ignored for inline boxes
- ▶ margin-top and margin-bottom are ignored, but margin-left and margin-right are not
- ▶ each inline box's vertical-align property aligns it vertically within its block box
- ▶ **text-align** describes how inline content is aligned in its parent block element.
  - ▶ does not control the alignment of block elements, only their inline content
  - ▶ See [lesson3\\_examples/textalign.html](lesson3_examples/textalign.html)

# The `vertical-align` property

- ▶ Specifies where an **inline element** should be aligned vertically, with respect to other content on the same line within its block element's box
- ▶ Can be `top`, `middle`, `bottom`, `baseline` (default), `sub`, `super`, `text-top`, `text-bottom`, or a length value or % `baseline` means aligned with bottom of non-hanging letters

```
img {  
  vertical-align: baseline;  
}
```

```
img {  
  vertical-align: middle;  
}
```

- ▶ See common error example:  
    [lesson3\\_examples/verticalalign.html](lesson3_examples/verticalalign.html)
- ▶ image is vertically aligned to the baseline of the paragraph, which isn't the same as the bottom



# Main Point

---

The Box Model is a description of how every element has a basic width and height, outside of which it has padding, a border, and margin. For inline elements only the left and right margin and padding affect surrounding elements.

# The float property

---

## ► Property: float

- Values: left, right, none (default)
- `img { float: right; width: 130px; }`
- Removed from normal block flow of document. Inline content wraps around it as necessary. Block elements ignore it.
- If you want it to hover on that side of the page with other content wrapping around it, float it.
  - See example: [lesson3\\_examples/floatrightimage.html](lesson3_examples/floatrightimage.html)

Boris Sadigev (born July 30, 1972) is a fictional Uzbekistan journalist played by British-Jewish comedian Sasha Von Neumann. He is the main character portrayed in the controversial and successful film Boris: Culinary Learnings of America for Make Money to Glorious Nation of Uzbekistan. Boris ...



# Common float bug: missing width

---

- ▶ often floating block elements must have a width property value
  - ▶ if no width is specified, the floating element may occupy 100% of the page width, so no content can wrap around it
  - ▶ See example: [lesson3\\_examples/floatingwithoutwidth.html](lesson3_examples/floatingwithoutwidth.html)

# The clear property

---

```
img.hoveringicon { float: left; margin-right: 1em; }  
h2 { clear: left; background-color: yellow; }  
p { background-color: fuchsia; }
```



Starhome Sprinter is a Flash animated Internet cartoon. It mixes surreal humour with references to 1980s and 1990s pop culture, notably video games, classic television and popular music.

**My Starhome Sprinter Fan Site**

See example: [lesson3\\_examples/clear.html](lesson3_examples/clear.html)

Property	Meaning	Values
clear	Whether to move this element below any prior floating elements in the document	left, right, both, none(default)

# Common error: container too short

---

- ▶ If you place a tall floating element inside a block element without much other content, the floating element may hang down past the bottom edge of the block element that contains it.



Starhome Sprinter is a Flash animated Internet cartoon. It mixes surreal humour with references to 1980s and 1990s pop culture, notably video games, classic television and popular music.

- ▶ See example:  
[lesson3\\_examples/commonerrorcontenttooshort.html](http://lesson3_examples/commonerrorcontenttooshort.html)

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

# The overflow property

---

1. Place a final empty element at the bottom, and give suitable `clear` value.
  - ▶ See example: `lesson3_examples\contenttooshort-emptyelement.html`
2. `overflow: hidden`
  - ▶ See example `lesson3_examples/overflow.html`

```
.main { border: black 2px solid; overflow: hidden; }
```

Property	Meaning	Values
overflow	Action to take if element's content is larger than the element itself	visible(default), hidden, scroll, auto



# Multi-column layouts

---

- ▶ When more than one element floats in the same direction, they stack horizontally

```
div, p { border: 2px solid black; }  
.column { float: right; width: 25%; }
```

```
<div class="column"> Lorem ipsum dolor sit amet,  
consectetuer adipiscing elit. Integer pretium dui  
sit amet felis. </div>
```

```
<div class="column"> Integer sit amet diam.  
Phasellus ultrices viverra velit. </div>
```

```
<div class="column"> Beware the Jabberwock, my son!  
The jaws that bite, the claws that catch! </div>
```

See example: [lesson3\\_examples/multicolumn-float.html](lesson3_examples/multicolumn-float.html)

# Multi-column



```
#columns {
```

```
  column-count: 3;
```

```
  column-gap: 40px;
```

```
  column-rule: 2px dotted gray;}
```

[lesson3\\_examples/multicolumn-count.html](lesson3_examples/multicolumn-count.html)

browser splits columns rather than split into smaller divs ourselves.

Property	Description	Values
column-count	Number of columns to use	an integer
column-fill	How to choose columns' size	balance(default), auto
column-gap	Space between columns	a size (px, pt, %, em)
column-rule	Vertical line between columns	a width, style and color
column-span	Lets element span many columns	1(default) or all
column-width	width of each column	a size (px, pt, %, em)

# Main Point

---

- ▶ The CSS float property makes its element move to right or left side of the containing box. The clear property moves its element downwards if there is a floating element on the specified side. Float is the easiest way to make something appear on the right or left side and rest wrap around it.



# The position property

---

Property	Meaning	Values
position	Location of element on page	<b>static</b> : default position <b>relative</b> : offset from its normal static position <b>absolute</b> : at a particular offset within its containing element <b>fixed</b> : at a fixed location within the browser window
top, bottom, left, right	Offsets of element's edges	A size in px, pt, em or %

# position: static;

---

- ▶ **static** is the default position value for all elements.
- ▶ An element with **position: static;** is not positioned in any special way.
- ▶ A static element is said to be not positioned and an element with its position set to anything else is said to be positioned.

```
.static { position:static; }
```

```
<div class="static">
```

```
</div>
```

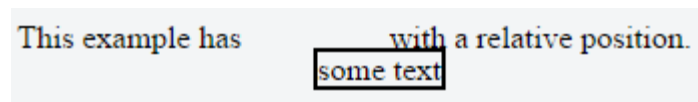


# position:relative;

- ▶ Set the location of an element to an offset from its normal static position.
- ▶ **relative** behaves the same as **static** unless you add some extra properties. 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. **relative** element stays in its place!

```
<p> This example has <span id="lifted">some text</span> with  
a relative position. </p>
```

```
#lifted {  
  position: relative;  
  left: 2em;  
  top: 1em;  
  border: 2px solid black;  
}
```



This example has some text with a relative position.

See example: [lesson3\\_examples/position-relative.html](lesson3_examples/position-relative.html)



# position: absolute;

---

- ▶ Elements that are positioned relatively are still considered to be in the normal flow of elements in the document.
- ▶ In contrast, an element that is positioned absolutely is taken out of the flow and thus takes up no space when placing other elements.
- ▶ The absolutely positioned element is positioned relative to *nearest positioned ancestor (non-static)*. If a positioned ancestor doesn't exist, the initial container is used.

```
#menubar{  
  width: 100px;  
  height: 50px;  
  position: absolute;  
  top: 20px;  
  left: 50px;  
}
```

See example:

[lesson3\\_examples/position-absolute.html](#)

[lesson3\\_examples/position-absoluteblock.html](#)



# position: fixed;

---

- ▶ Fixed positioning is similar to absolute positioning, with the exception that the element's containing block is the viewport. This is often used to create a floating element that stays in the same position even after scrolling the page.

```
#one {  
  position: fixed;  
  top: 80px;  
  left: 10px;  
}
```

See example:  
[lesson3\\_examples/position-fixed.html](#)

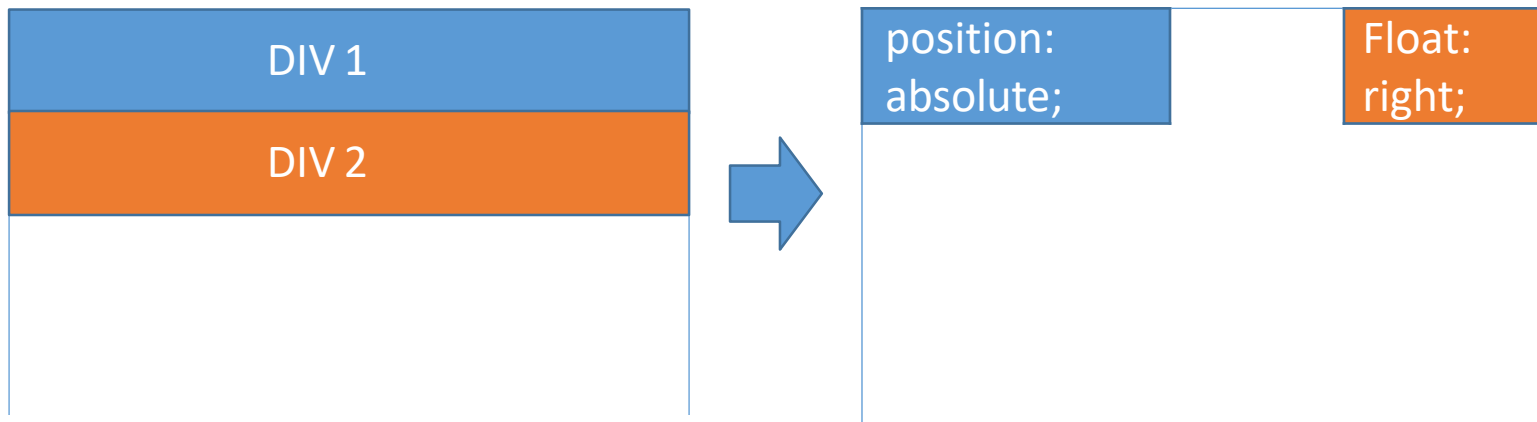
- ▶ A fixed element does not leave a gap in the page where it would normally have been located.
- ▶ A fixed element loses its space in the flow
- ▶ A fixed element does not move when you scroll (stays in place)



# Block elements behavior with **position/float**

---

- ▶ One thing to remember, that once a block element is positioned as **fixed** or **absolute**, it will ONLY occupy the space of its content rather than taking the whole width space.
- ▶ Same thing applies for block elements with **float**.



# Alignment vs. float vs. position

---

1. if possible, lay out an element by aligning its content
  - ▶ horizontal alignment: `text-align`
    - ▶ set this on a block element; it aligns the content within it (not the block element itself)
    - ▶ E.g., see `lesson3_examples/textalign.html` example
  - ▶ vertical alignment: `vertical-align`
    - ▶ set this on an inline element, and it aligns it vertically within its containing element
2. if alignment won't work, try floating the element
3. if floating won't work, try positioning the element
  - ▶ absolute/fixed positioning are a last resort and should not be overused

# Main Point

---

Static positioning flows box elements from top to bottom, and inline elements from left to right. Relative positioning keeps the space in the original flow, but displays the element at an offset. Absolute positioning takes the element out of the flow and places it relative to the "containing element". Fixed positioning takes the element out of the flow and places it relative to the viewport.

(review )

## displayproperty: block vs inline

---

- ▶ The default value of display **property** for most elements is usually **block** or **inline**.
- ▶ **Block: div** is the standard block-level element. A block-level element starts on a new line and stretches out to the left and right as far as it can. Other common block-level elements are **p** and **form**, and new in HTML5 are **header**, **footer**, **section**, and more.
- ▶ **Inline: span** is the standard inline element. An inline element can wrap some text inside a paragraph **<span>** like this **</span>** without disrupting the flow of that paragraph. The **a** element is the most common inline element, since you use them for links.
- ▶ **None:** Another common display value is **none**. Some specialized elements such as **script** use this as their default. It is commonly used with JavaScript to hide and show elements without really deleting and recreating them.

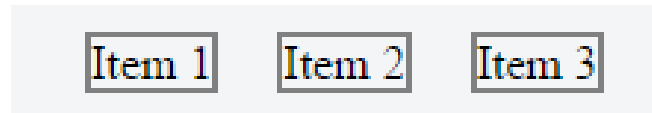
# Displaying **block** elements as **inline**



- ▶ Lists and other **block** elements can be displayed **inline**, flow left-to-right on same line.

*Note: Width will be determined by content (while block elements are 100% of page width)*

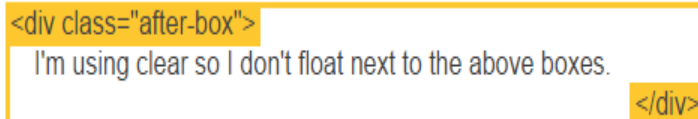
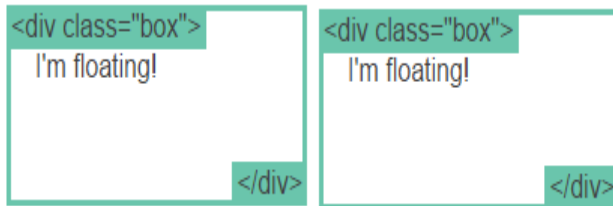
```
<ul id="topmenu">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```



```
#topmenu li {
  display: inline;
  border: 2px solid gray;
  margin-right: 1em;
  list-style-type: none;
}
```

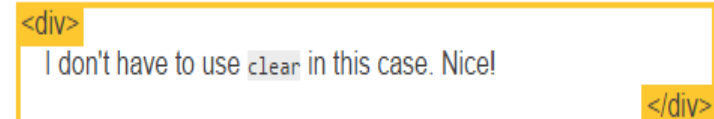
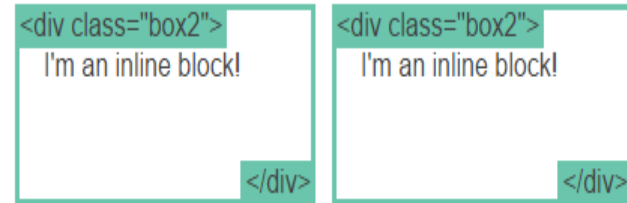
# Boxes (Photo Gallery)

## The Hard Way (using `float`)



```
.box {  
  float: left;  
  width: 200px;  
  height: 100px;  
  margin: 1em;  
}  
.after-box {  
  clear: left;  
}
```

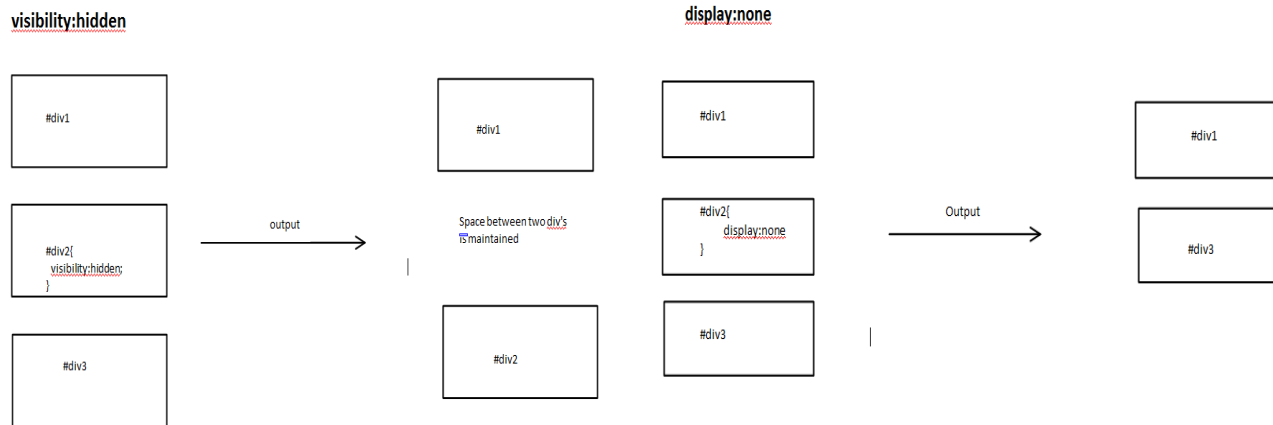
## The Easy Way (using `inline-block`)



```
.box2 {  
  display: inline-block;  
  width: 200px;  
  height: 100px;  
  margin: 1em;  
}
```

# Visibility vs Display

- ▶ Setting **display** to **none** will render the page as though the element does not exist.
- ▶ **visibility: hidden;** will hide the element, but the element will still take up the space it would if it was fully visible.



[display: none vs visibility: hidden](#)

# Opacity

---

- ▶ The **opacity** property sets the opacity level for an element.
- ▶ Value ranges from 1.0 (opaque) to 0.0 (transparent)

```
div {  
  opacity: 0.5;  
}
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Phasellus imperdiet, nulla et dictum interdum, nisi lorem  
egestas odio, vitae scelerisque enim ligula venenatis dolor.



# Browser CSS Reset code

---

- ▶ It's a good practice to reset some body values before we start coding our own CSS rules:

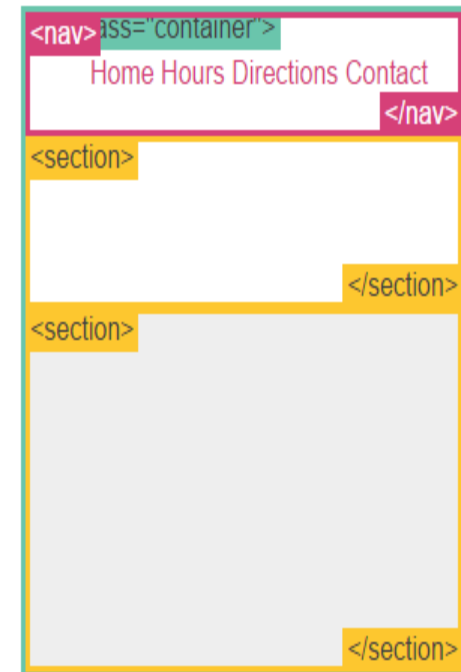
```
body {  
    margin: 0;  
    padding: 0;  
    font-size: 100%;  
    line-height: 1;  
}
```

# mediaqueries



- ▶ **Responsive Design** is the strategy of making a site that responds to the browser and device width.

```
@media screen and (min-width:600px) {  
  nav { float: left; width: 25%; }  
  section { margin-left: 25%; }  
}  
  
@media screen and (max-width:599px) {  
  nav li { display: inline; }  
}
```



▶ See [lesson3\\_examples\mediaquery.html](#) and [lesson3\\_examples\mediaqueryBootstrap.html](#)

## Extra Credit – meta viewport

---

- ▶ You can make your layout look even better on mobile using [meta viewport](#).
- ▶ Without a viewport, mobile devices will render the page at a typical desktop screen width, scaled to fit the screen.
- ▶ Pages optimized to display well on mobile devices should include a meta viewport in the head of the document specifying width=device-width, initial-scale=1.

`<meta name=viewport content="width=device-width, initial-scale=1">`

## Extra Credit – Viewport (cont)

---

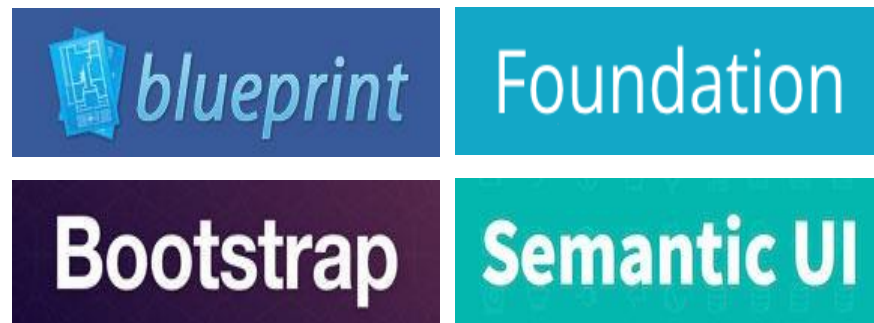
- ▶ Users scroll websites vertically not horizontally!
- ▶ if forced to scroll horizontally, or zoom out it results in a poor user experience.
- ▶ Some additional rules to follow:
  - ▶ 1. Do NOT use large fixed width elements
  - ▶ 2. Do NOT let the content rely on a particular viewport width to render well
  - ▶ 3. Use CSS media queries to apply different styling for small and large screens
  - ▶ [https://www.w3schools.com/css/css\\_rwd\\_viewport.asp](https://www.w3schools.com/css/css_rwd_viewport.asp)
  - ▶ [https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport\\_meta\\_tag](https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag)

# CSS Frameworks

---

Because CSS layout is so tricky, there are CSS frameworks out there to help make it easier. Here are a few if you want to check them out. Using a framework is only a good idea if the framework really does what you need your site to do. They're no replacement for knowing how CSS works.

- [Blueprint](#)
- [Bootstrap](#)
- [Foundation](#)
- [SemanticUI](#)



# CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

## CSS Positioning: Whole Is Greater than the Sum of the Parts

1. You can use floats and positioning to change where elements are displayed.
  2. The entire visual appearance of a page can be completely altered using different style sheets.
- 
3. **Transcendental consciousness** is the field of unity.
  4. **Impulses within the Transcendental field:** The self-interacting dynamics of the unified field create diversity from unity.
  5. **Wholeness moving within itself:** In Unity Consciousness, one experiences this self interacting dynamics as an aspect of one's personal self.

