

# Lecture 3: Layout

**CS472 Web Programming**

**Maharishi University of Management**

**Department of Computer Science**

**Assistant Professor Obinna Kalu, September 2017**

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. Layout contents and examples credit goes to [learnlayout.com](http://learnlayout.com), Greg Smith and Isaac Durazo.

## **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.

# Page Layout

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. It is this relationship that becomes the tricky part. The question is always: "What is this positioned to".

**Science of Consciousness:** The whole is greater than the sum of its parts.

Why it's important to learn Layout?

# Sections of a page: `<div>` vs `<span>`

- `<div>` is a block element
- `<span>` is an inline element
- They have no onscreen appearance, but you can apply styles to them
- They carry no significant semantic meaning



```
<div class="shout">
  <h2>Hello</h2>
  <p class="special">See our specials!</p>
  <p>We'll beat <span class="shout">all prices!</span></p>
</div>
```

# CSS context selectors



`selector1 selector2 { properties }`

Applies the given properties to **selector2 only** if it is inside a selector1 on the page



`selector1 > selector2 { properties }`

Applies the given properties to **selector2 only** if it is direct child of selector1 (in the DOM)

# Example



```
<div id="ad">
  <p>Shop at <strong>Hardwick's Hardware</strong></p>
  <ul>
    <li class="line"><em>The</em><strong>best</strong> prices!</li>
    <li><em><strong>Act while supplies last!</strong></em></li>
  </ul>
</div>
```

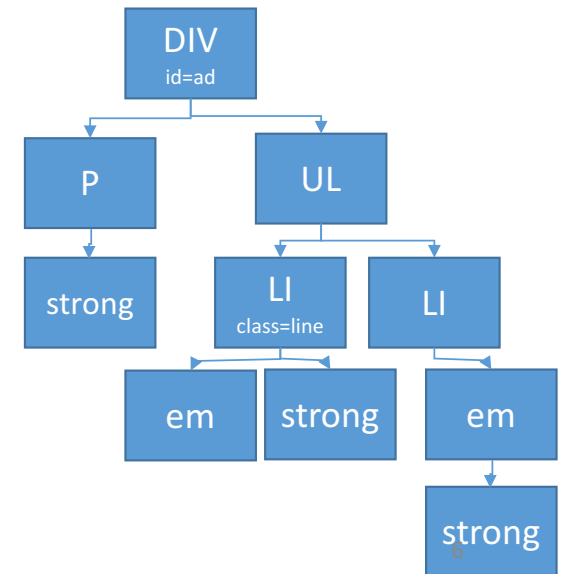


```
ul > li { background-color: blue; }
li strong { color: red; }
li > strong { color: green; }
#ad li.line strong { text-decoration: underline; }
```



Shop at **Hardwick's Hardware**

- The best prices!
- **Act while supplies last!**



# Main Point

The <div> tag provides a generic block level element that can be used for any division or section of your page. The <span> tag provides a generic inline element for specifying any range of text inside a box. By using these tags, combined with CSS context selectors (direct child or ascendant) we can make write CSS rules that are general enough to be re-used, while still being specific enough not to create confusion. **Science of Consciousness:** TM allows our mind to transcend the finest levels of awareness while our body remains in the relative, providing a perspective on the coexistence of opposites.

# 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.



# CSS properties for borders

`border: border-width border-style border-color;`



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

border-color, border-width, border-style, border-bottom, border-left, border-right, border-top, 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

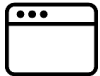


```
h2 {  
  border-left: thick dotted #CC0088;  
  border-bottom-color: rgb(0, 128, 128);  
  border-bottom-style: double;  
}
```

# Rounded corners with `border-radius`



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



Text Paragraph

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

# Styling tables

All standard CSS styles can be applied to a table, row, or cell



```
table, td, th {  
    border: 2px solid black;  
}  
table {  
    border-collapse: collapse;  
}  
td {  
    background-color: yellow;  
    text-align: center;  
    width: 30%;  
}
```

Column 1	Column 2
1,1	1,2
2,1	2,2

Without border-collapse

Column 1	Column 2
1,1	1,2
2,1	2,2

With border-collapse

By default, the overall table has a separate border from each cell inside, the **border-collapse** property merges these borders into one.

# Dimensions

For **Block elements 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; }  
h2 { width: 50%; }
```

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.

# The CSS Box Model

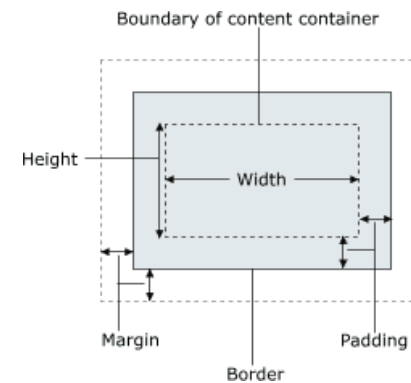
For layout purposes, every element is composed of:

- 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)

Visual width = content width + L/R padding + L/R border + L/R margin

Visual height = content height + T/B padding + T/B border + T/B margin

The standard `width` and `height` properties refer ONLY to the content's width and height.



# 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;
}
```







# Centering a block element: **auto** margins

- Works only if width is set (otherwise, it will occupy entire width of page)
- To center inline elements within a block element, use **text-align: center;**




```
p {  
    margin-left: auto;  
    margin-right: auto;  
    width: 750px;  
}  
OR  
p {  
    margin: auto;  
    width: 750px;  
}
```


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.

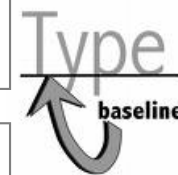
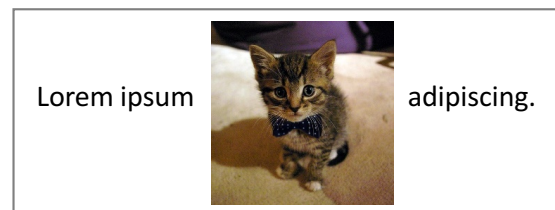
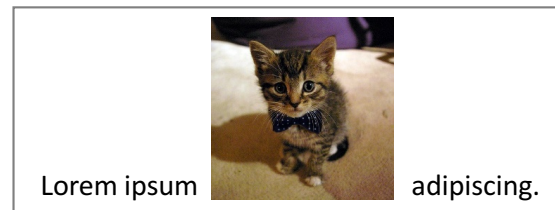
# The `vertical-align` property

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;  
}
```



# Block Elements in details

- By default **block** elements take the entire width space of the page unless we specify.
- **Margin** and **Padding** work as expected
- 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** element within the page.

## **inline** Elements in details

- Size properties (**width**, **height**, **min-width**, etc.) are ignored for inline elements.
- **margin-top** and **margin-bottom** are ignored, but **margin-left** and **margin-right** are not
- The containing block element's **text-align** property controls horizontal position of inline elements within it
- Each inline element's **vertical-align** property aligns it vertically within its block element

# 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;
}
```

# 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. You can use web development tools like firebug to inspect the box model for each element. *Harmony exists in diversity.*

# The `display` property

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.

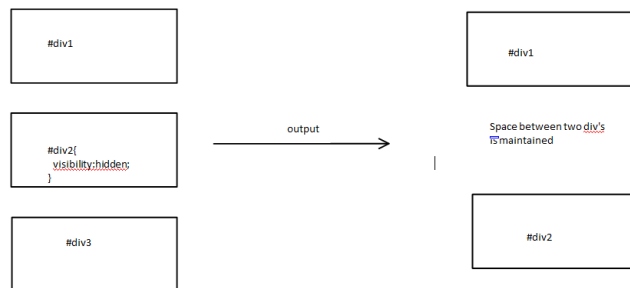


# Visibility vs Display

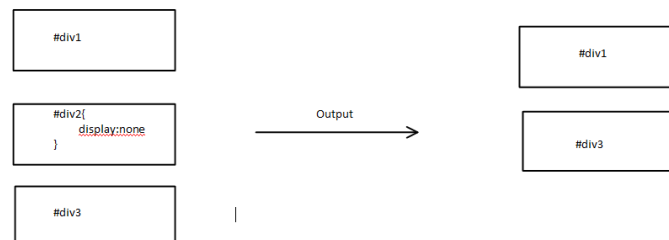
Setting **display** to **none** will render the page as though the element does not exist.

**visibility: hidden;** will hide the element, but the element will still take up the space it would if it was fully visible.

visibility:hidden



display:none



<http://www.kodecrash.com/>

# 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;

**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!



```
.relative1 {  
    position: relative;  
}  
.relative2 {  
    position: relative;  
    top: 20px;  
    left: 20px;  
    width: 500px;  
}
```



# position: fixed;

A **fixed** element is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. As with relative, the **top**, **right**, **bottom**, and **left** properties are used.



```
.fixed {  
    position: fixed;  
    bottom: 0;  
    right: 0;  
    width: 200px;  
}
```

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)

```
<div class="fixed">  
    Hello! Don't pay  
    attention to me yet.  
</div>
```

## **position: absolute;**

**absolute** behaves like **fixed** except relative to the nearest positioned ancestor instead of relative to the viewport and when you scroll the page, it moves.

**absolute** element loses its space in the flow (surrounding contents will take its place)

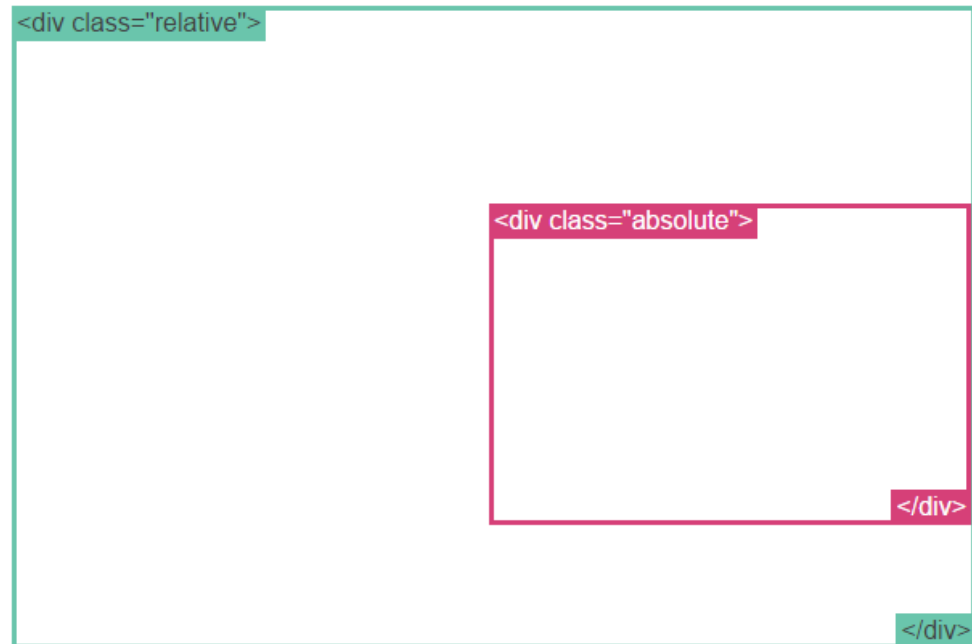
If an absolutely-positioned element has no positioned ancestors, it uses the document body, and still moves along with page scrolling.

Remember, a "positioned" element is one whose position is anything except **static**.

# Absolute example

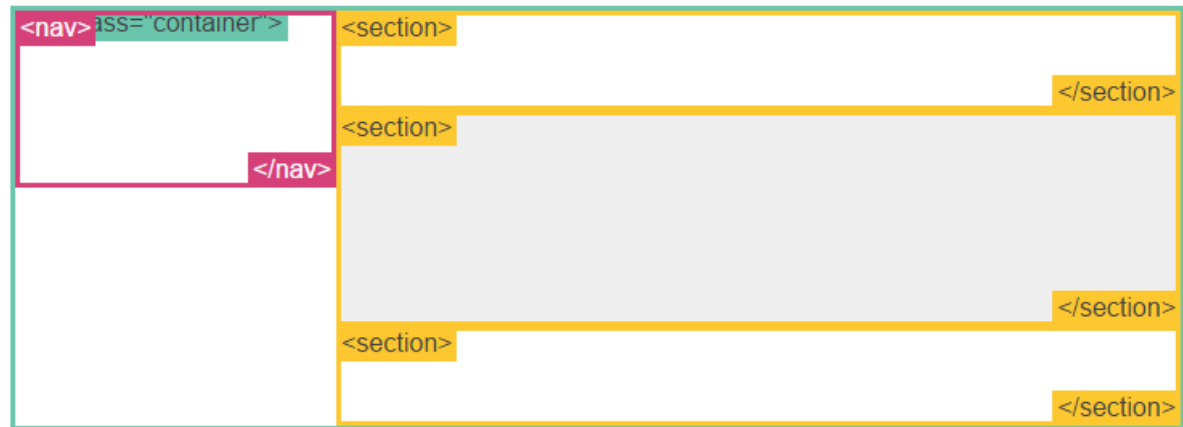


```
.relative {  
  position: relative;  
  width: 600px;  
  height: 400px;  
}  
.absolute {  
  position: absolute;  
  top: 120px;  
  right: 0;  
  width: 300px;  
  height: 200px;  
}
```



# Position Layout Example

```
.container {  
  position: relative;  
}  
nav {  
  position: absolute;  
  left: 0px;  
  width: 200px; }  
section {  
  margin-left: 200px;  
}
```



# float

Float is intended for wrapping text around images.

```
img {  
  float: right;  
  margin: 0 0 1em 1em;  
}
```

- When we float a `div` element it will comply to `width` and `height` properties rather than behaving like a block element.
- A floating element is removed from normal document flow.
- Underlying text wraps around it as necessary.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis.



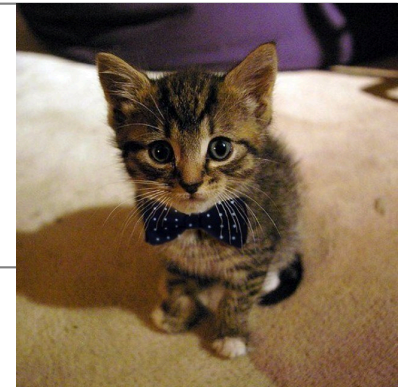


# The clearfix hack

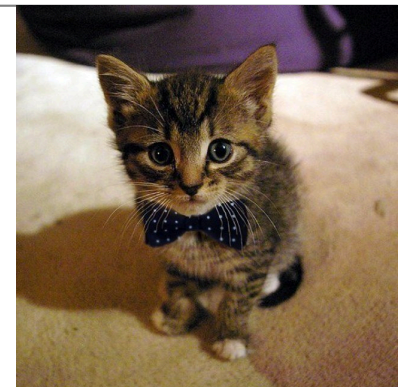
Here is a weird, bad thing that can sometimes happen when using floats:

```
img {  
  float: right;  
}  
  
.clearfix {  
  overflow: auto;  
}
```

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



```
<div class="clearfix">  
  Lorem ipsum dolor sit amet, consectetur  
  adipiscing elit. Phasellus imperdiet, nulla et  
  dictum interdum, nisi lorem egestas odio,  
  vitae scelerisque enim ligula venenatis  
  dolor.
```

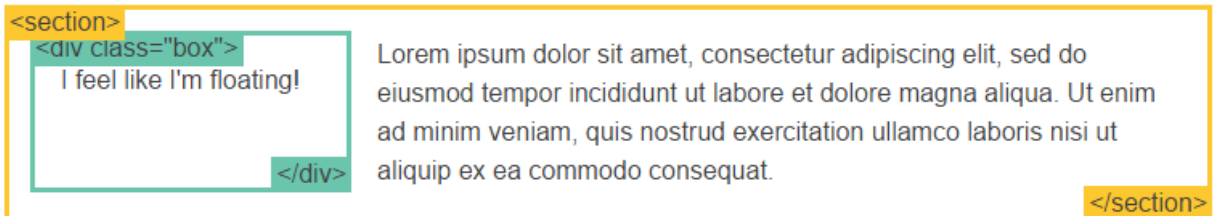


# clear

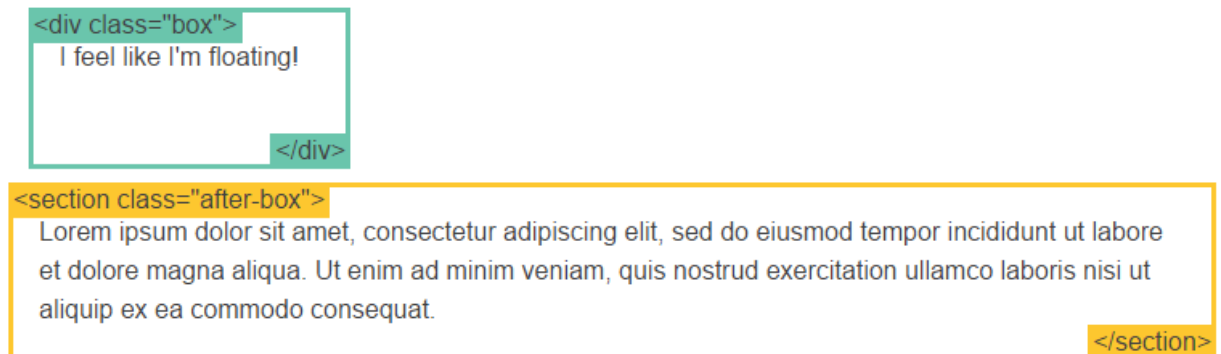
```
<div class="box">...</div>  
<section>...</section>
```

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

In this case, the section element is actually after the **div**. However, since the **div** is floated to the left, this is what happens: the text in the section is floated around the div and the section surrounds the whole thing. What if we wanted the section to actually appear after the floated element?

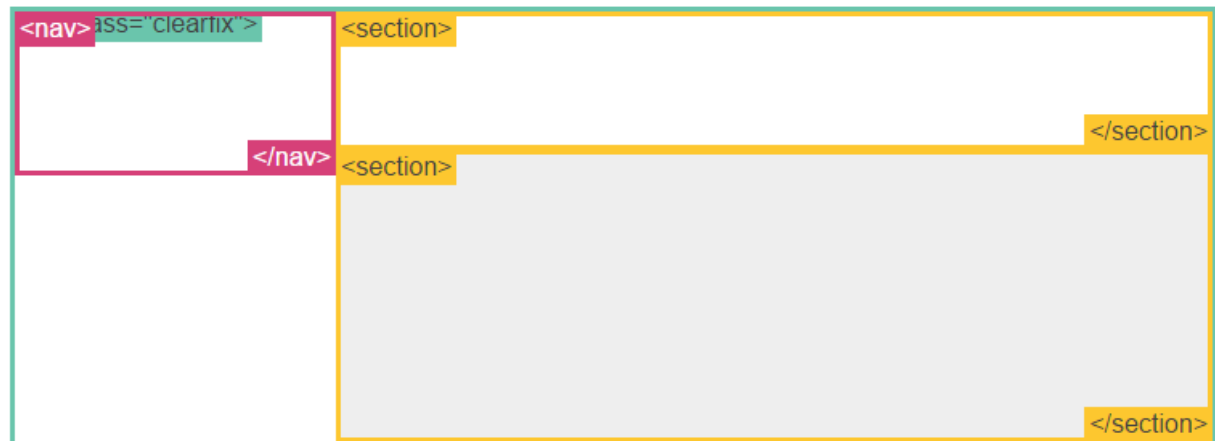


Using **clear** we have now moved this section down below the floated **div**. You use the value **left** to clear elements floated to the **left**. You can also clear **right** and **both**.



# Float Layout Example

```
nav {  
  float: left;  
  width: 200px;  
}  
section {  
  margin-left: 200px;  
}
```

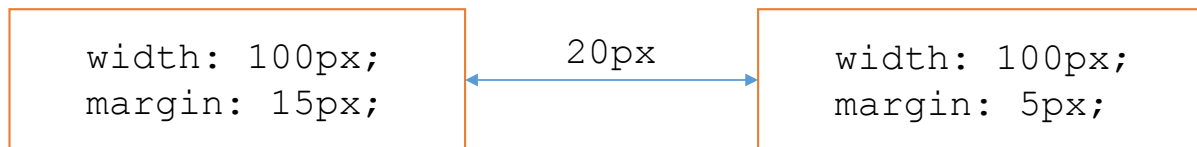


This example works just like the last one. Notice we put a **clearfix** on the container. It's not needed in this example, but it would be if the **nav** was longer than the non-floated content.

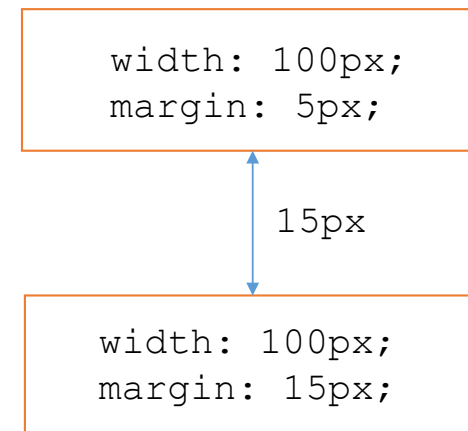
```
.clearfix {  
  overflow: auto;  
}
```

# Margin Collapse

Margin is usually added up when calculating the overall space any element would occupy.



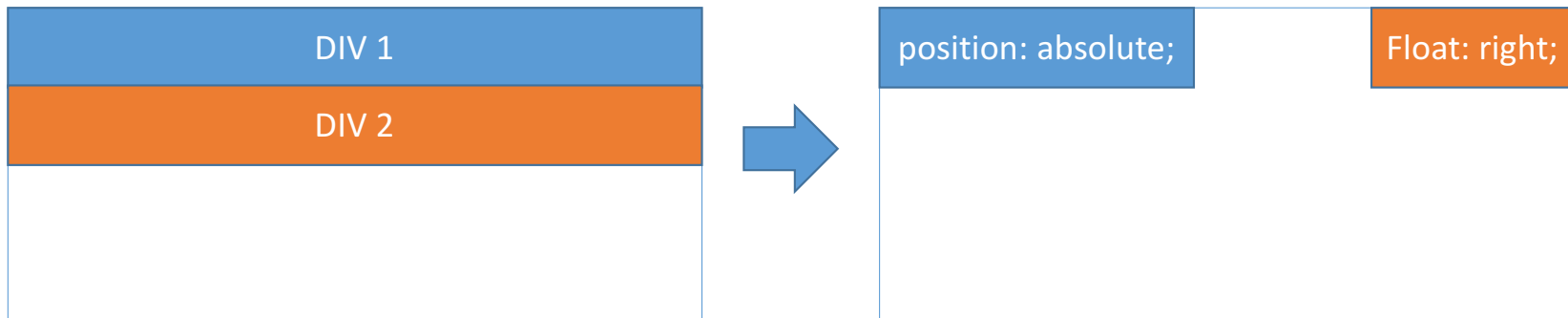
In certain cases (check links) only border-top and border-bottom will share the bigger margin space (margin collapse).



<http://www.howtocreate.co.uk/tutorials/css/margincollapsing>  
[https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Box\\_Model/Mastering\\_margin\\_collapsing](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Mastering_margin_collapsing)  
<https://css-tricks.com/what-you-should-know-about-collapsing-margins/>

## 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**.

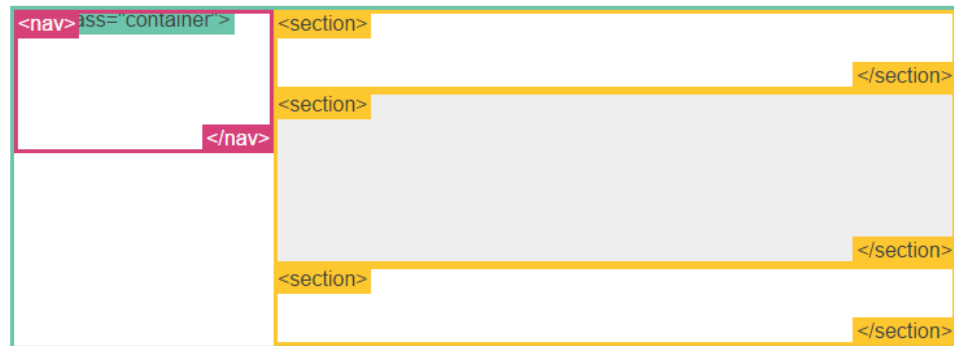


# Fluid Layout

You can use percent for layout, but this requires more work.

When this layout is too narrow, the **nav** gets squished. Worse, you can't use **min-width** on the **nav** to fix it, because the right column wouldn't respect it.

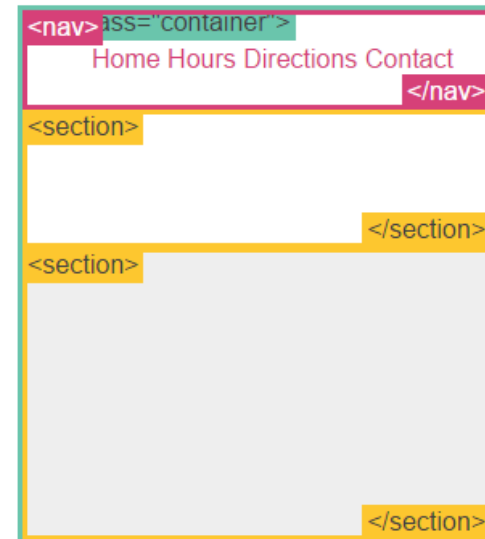
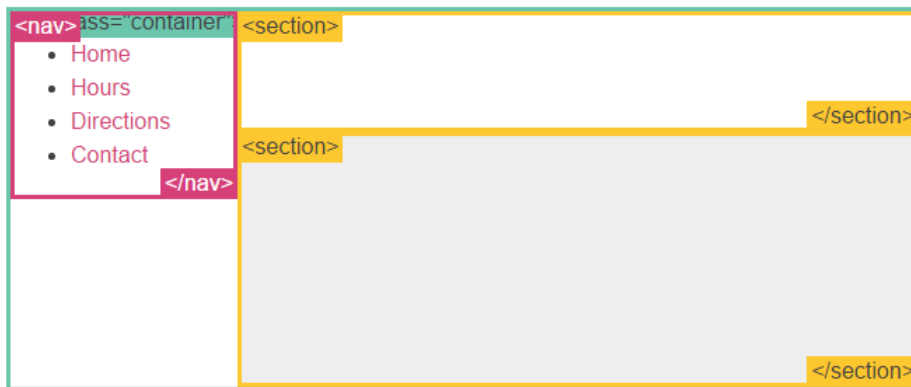
```
nav {  
  float: left;  
  width: 25%;  
}  
section {  
  margin-left: 25%;  
}
```



# media queries

**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; }  
}
```



# Extra Credit - Reading

You can make your layout look even better on mobile using [meta viewport](#).

## An Introduction to Meta Viewport and @viewport

### Introduction

Support for the viewport meta tag in Opera's mobile products has been around for quite some time, and from Opera Mobile 11 onward, we have made our viewport implementation more robust, added support for new mechanisms to deal with different screen densities, and included an implementation of our own [CSS @viewport rule proposal](#). In addition, Opera Mini 6 and later now also comes with basic support for the viewport meta tag.

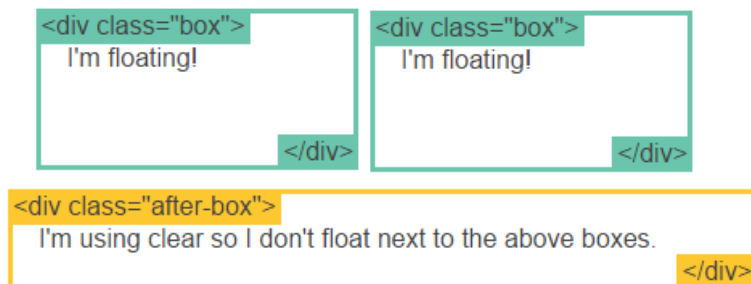
Update 27 May 2012: There is experimental [support for -ms-viewport in Internet Explorer 10](#).

So, what better time than now to give you an introduction to the various viewport-related mechanisms you can use to optimize your site for mobile? Let's get started!



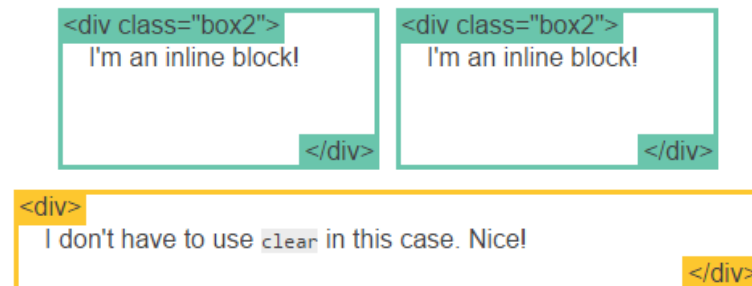
# Boxes (Photo Gallery)

## 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;
}
```

# Columns

This CSS set of properties let you easily make multi-column text.

```
.three-column {  
  padding: 1em;  
  column-count: 3;  
  column-gap: 1em;  
}
```

<div class="three-column">

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

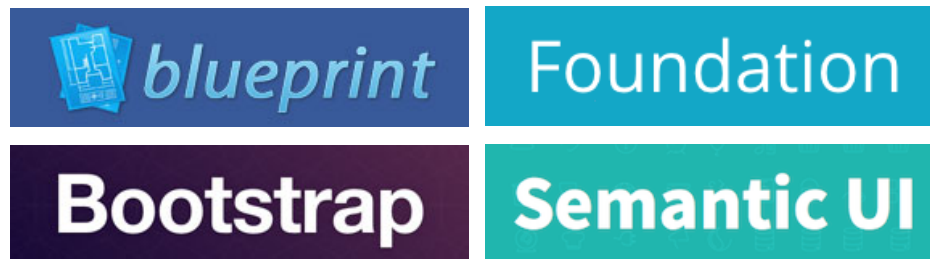
Lorem ipsum dolor sit  
amet, consectetur  
adipiscing elit.  
Phasellus imperdiet,  
nulla et dictum  
interdum, nisi lorem  
egestas odio, vitae  
scelerisque enim  
ligula venenatis dolor.

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

# 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](#)



# 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 browser window. **Science of Consciousness:** Transcendental consciousness is when our mind goes outside the window of the relative.

# Positions Review

Relative	Fixed	Absolute
Keep in its original place	Leave the flow (other content will take its place)	Leave the flow
Stay in its original place unless I specify Top, Right, Bottom, Left	Move with Top, Right, Bottom, Left based on the Viewport	Move with Top, Right, Bottom, Left based on the nearest positioned ancestor (not static) – if not found: body
Width still takes whole screen	Width is no more takes whole of screen	Width is no more takes whole of screen

**Block:** By default takes whole screen width, height of its content – width and height can be changed

**Inline:** By default takes width and height of its content – width and height cannot be changed

# Float Review

1. Leave the flow
2. `Width` is no more takes whole of screen
3. Sibling content of a floated element will wrap around it
4. Sequential floated element on the same direction align next to each other but their margin will be determined by the higher value
5. Any element that comes after a floated element should be `cleared` or it will overlap with the floated element (remember point 1)
6. A container of a floated element will not expand to the height of its floated element (remember point 1) to fix that we give it:  
`overflow: auto.`

# 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;  
}
```

# CSS Grid Layout

This is new way of implementing layout using CSS (Not yet supported by Microsoft Edge or IE).

```
div.container{  
  display: grid;  
  grid-template-columns: 30% 50% 20%;  
  grid-gap: 1em;  
}
```



# CSS Flexbox Layout

This is another way of implementing layout using CSS.

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

# CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

## *CSS Positioning*

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.
- 

1. **Transcendental consciousness** allows us to experience the unity within diversity
2. **Impulses within the Transcendental field:** The self interacting dynamics of the unified field create diversity from unity
3. **Wholeness moving within itself:** In Unity Consciousness, one experiences this self interacting dynamics as an aspect of one's person self.

