

# Computer Science Principles

## Web Programming

### Web-Presentation Design with CSS

CHAPTER 15: LAYOUT MANAGEMENT I (POSITIONING)

DR. ERIC CHOU

IEEE SENIOR MEMBER



# CSS3

---

Chapter 11: CSS Hierarchy and Selectors

Chapter 12: Text, Image and Foreground (Contents)

Chapter 13: Color and Background (Contents)

Chapter 14: Box Model (Padding, Border, and Margin)

**Chapter 15: Layout Management (Floating and Positioning: where should the Element go)**

Chapter 16: Layout Management (Page Level Planning)

Chapter 17: Layout Management (Transition, Transforms, and Animation: space and time domain transformation)

Chapter 18: CSS Techniques (Put Everything Together)



# Flow

---

LECTURE 1

# Normal Flow

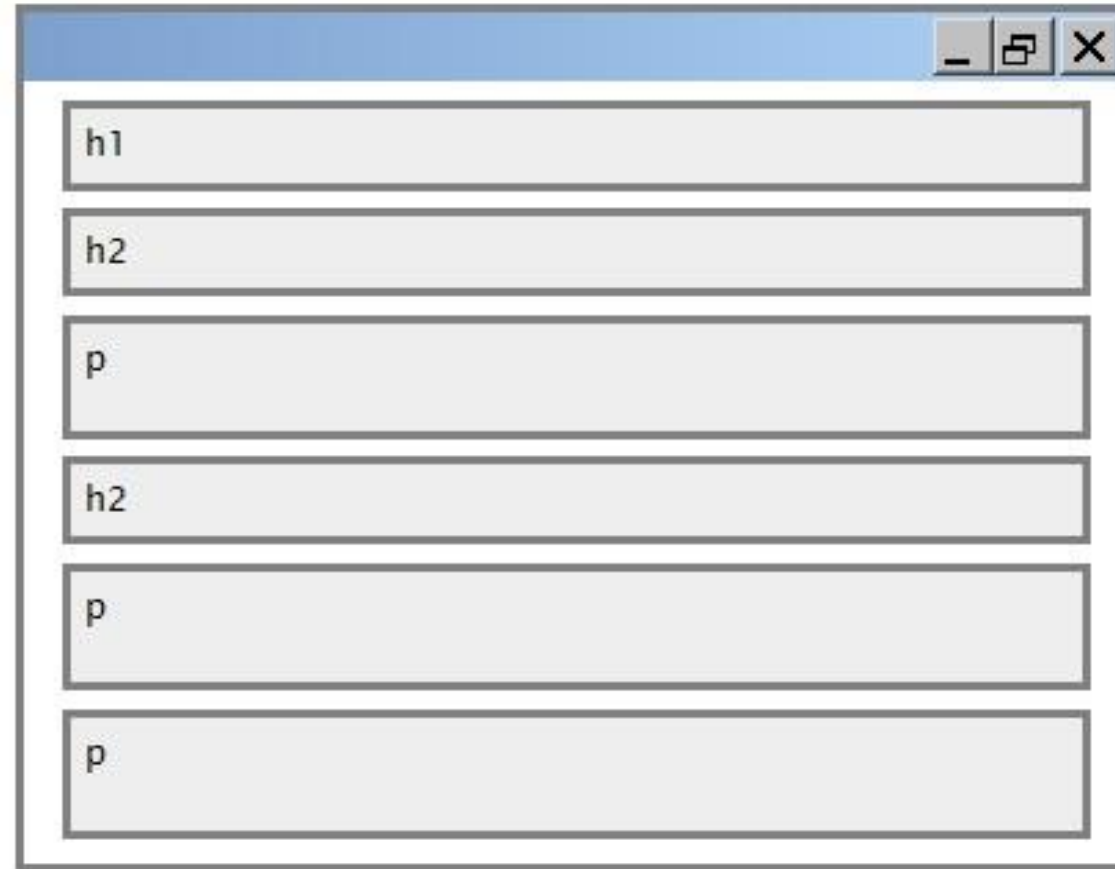
(Layout without CSS Setting)

---

- In the CSS layout model, text elements are laid out from top to bottom, in the order in which they appear in the source, and from left to right (in the left to right reading languages)
- **Block** elements stack up on the top of one another and fill the available width of the browser window or other containing element.
- **Inline** elements and text characters line up next to one another to fill the block elements.
- When the window or containing element is resized, the block elements expand or contract to the new width, and the inline content reflows to fit.

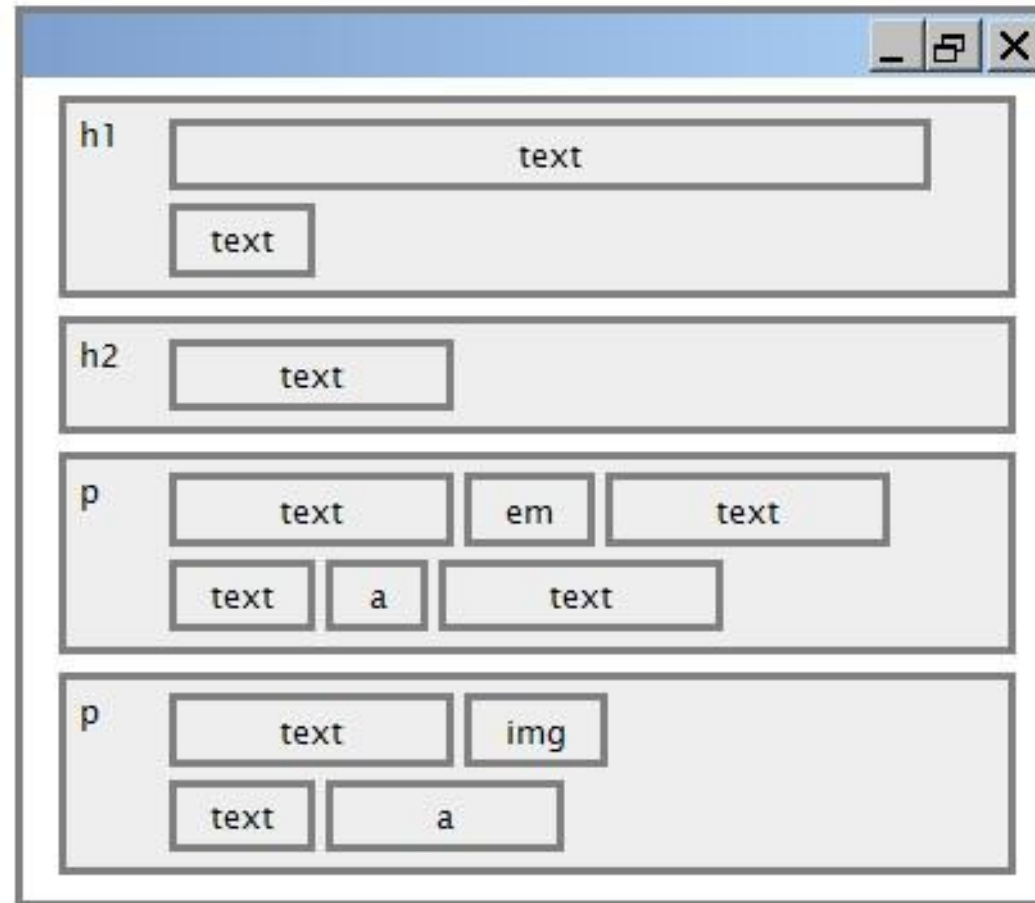
# Normal Flow

---



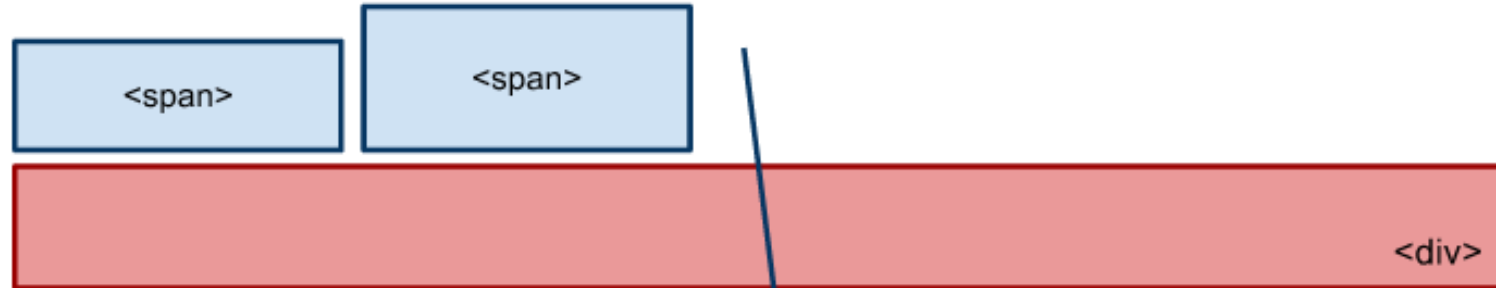
# Document Flow:

## Block and Inline Elements



spans are only as wide as the content they wrap.

Spans can be put inside divs. Note that regardless of how much horizontal space they take, the div always expands to the width of its parent



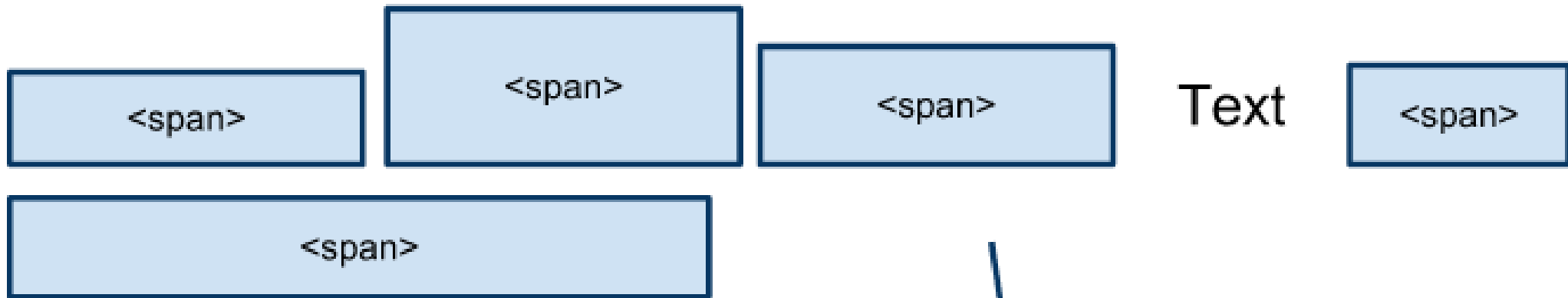
Divs always cause a new line

`<div>`  
always  
start a  
new line

---

# <span> automatically wrap

---



Note how spans flow with the text of the page. When they reach the end of the page, they automatically wrap.





# Floating

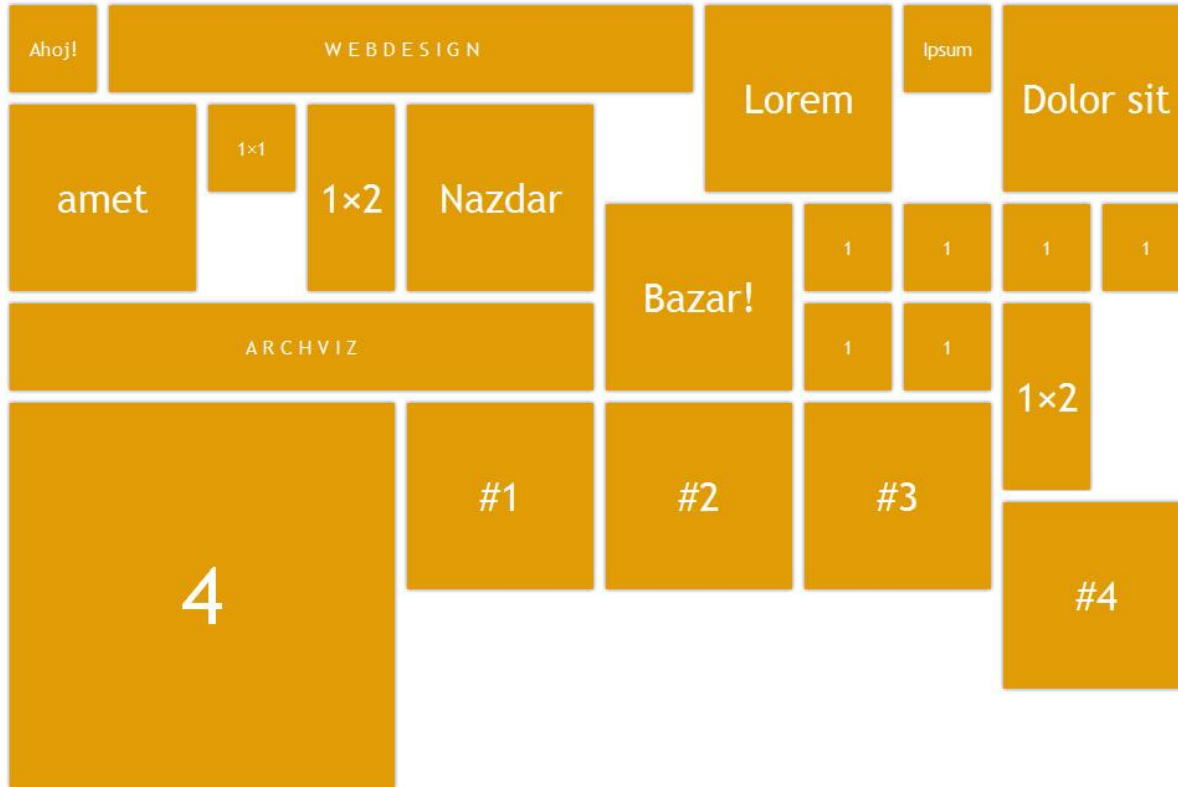
---

LECTURE 2



Floating  
move an  
element as  
far as possible  
to the left or  
right

---

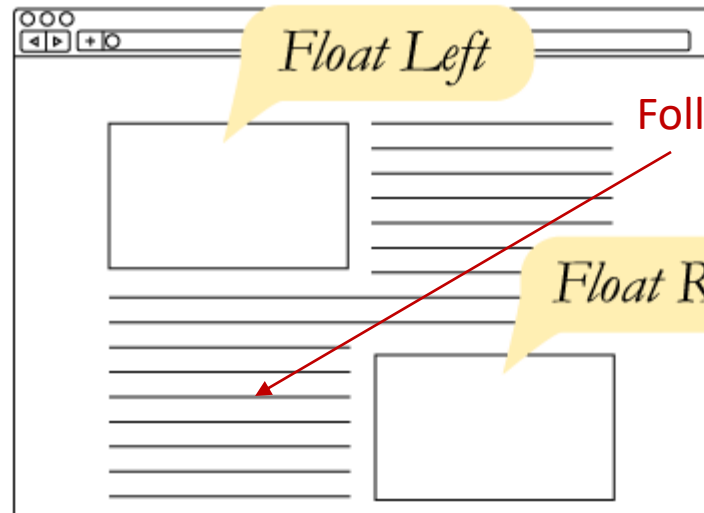


Floating  
move an  
element as  
far as possible  
to the left or  
right

---

# float

float: left | right | none | inherit



Web Layout

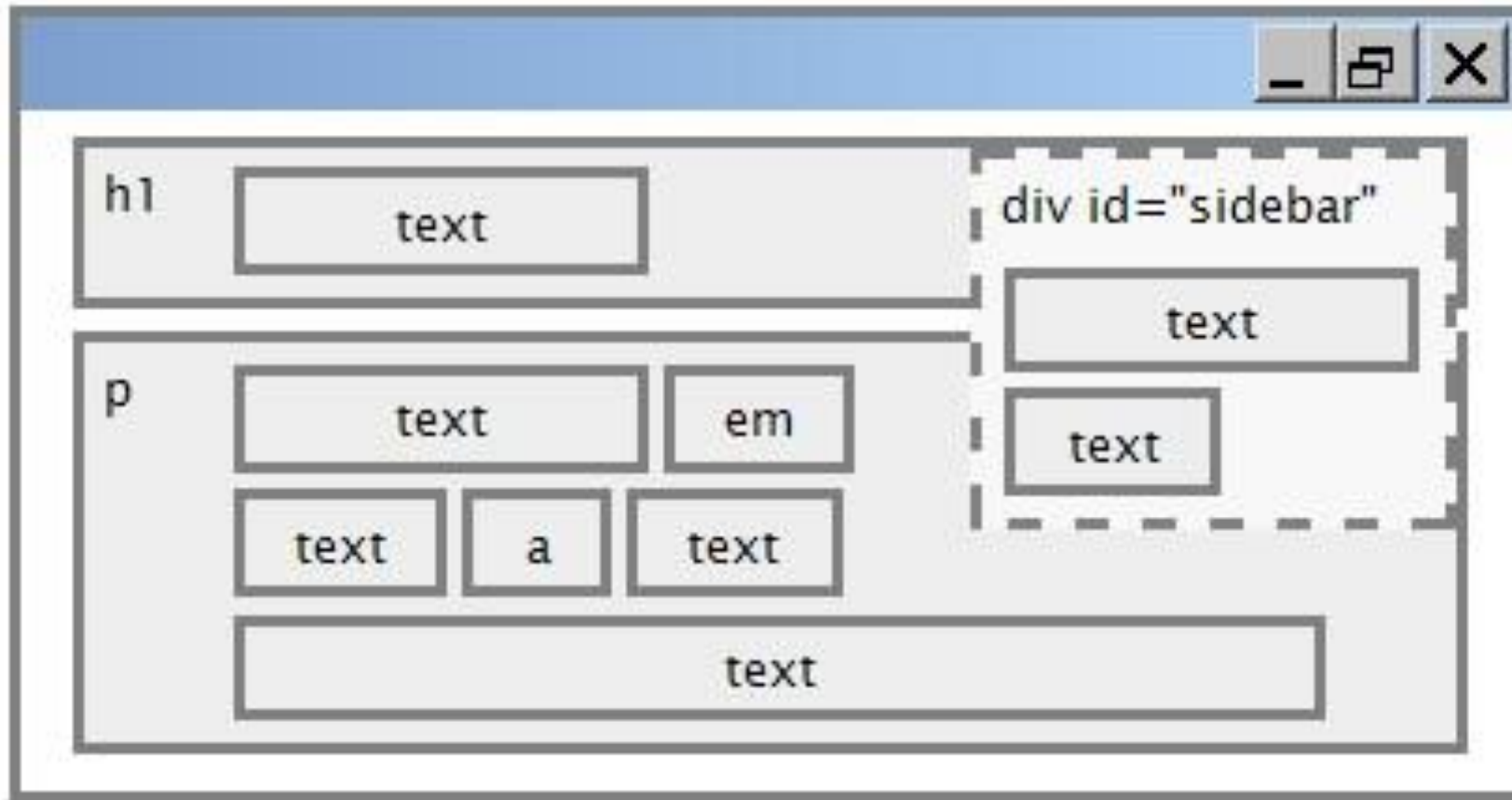
Following text flows around the previous floating elements.

Float Right

- a **floating** element is removed from normal document flow
- underlying text wraps around it as necessary

property	description
float	side to hover on; can be left, right, or none (default)

# Floating Out of Normal Flow



# Floating inline and block elements

none 1 before

none 2 before

left #1 left #2 none 1 after

right #2 right #1

none 2 after

- using Firebug, toggle the above `div`s from being aligned to floated...

I am not floating, no width set

I am floating right, no width set

I am floating right, no width set, but my text is very long so this paragraph doesn't really seem like it's floating at all, darn

I am not floating, 45% width

I am floating right, 45% width

- often floating elements should have a `width` property value
  - if no `width` is specified, other content may be unable to wrap around the floating element

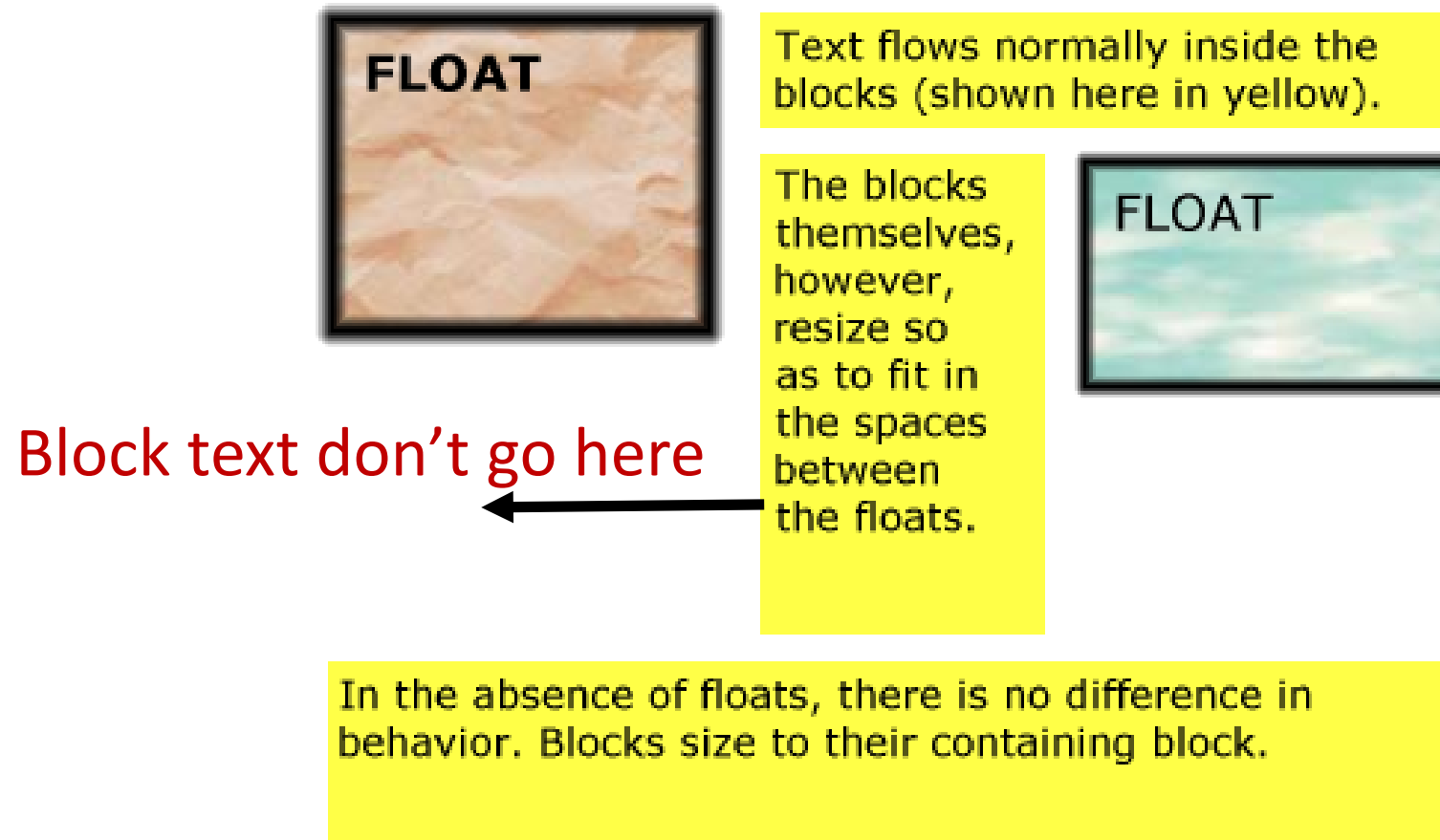
# Guidelines for Floating Inline Elements

[study the example on figure 15-4](#)

---

- Always provide a width for floated text elements.
- Floated inline elements behave as block elements.
- Margins on floated elements do not collapse.

# Floating block Elements





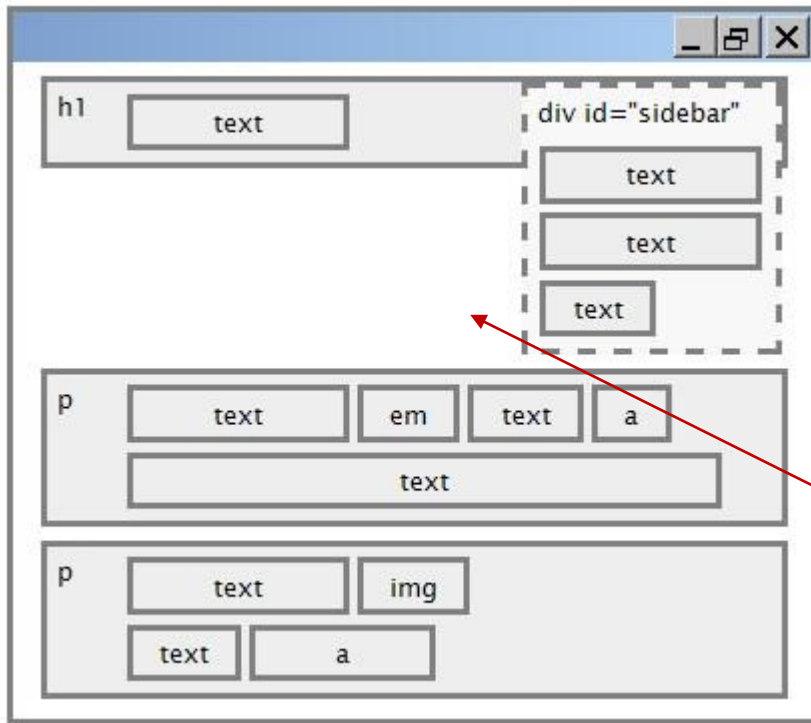
# Guidelines for Floating Block Elements

[study the example on figure 15-5](#)

---

- You must provide a width for floated block elements.
- Elements do not float higher than their reference in the source.

# Clearing floated elements study 15-6




property	description
<code>clear</code>	disallows floating elements from overlapping this element; can be <code>left</code> , <code>right</code> , <code>both</code> , or <code>none</code> (default)

Clearing this space.

```
div#sidebar { float: right; }
p { clear: right; }
```

CSS

# Floating right example (overflow)

<pre>&lt;p&gt;&lt;img src="images/homestar_runner.png" alt="homestar runner" /&gt;</pre>	HTML
<pre>Homestar Runner is a Flash animated Internet cartoon.</pre>	
<pre>It mixes surreal humour with ....&lt;/p&gt;</pre>	
<pre>p { border: 2px dashed black; }</pre>	
<pre>img { float: right; }</pre>	CSS
<p>Homestar Runner is a Flash animated Internet cartoon. It mixes surreal humour with ....</p>	
	

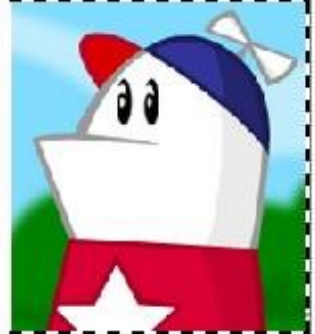
- We want the p containing the image to extend downward so that its border encloses the entire image

# Overflow property

```
p { border: 2px dashed black; overflow: hidden; }
```

CSS

Homestar Runner is a Flash animated Internet cartoon. It mixes surreal humour with ....



property	description
overflow	specifies what to do if an element's content is too large; can be auto, visible, hidden, or scroll

# Multiple Floating

P

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a mauris at ligula condimentum semper. Integer purus ante, sollicitudin sit amet consequat eget, vestibulum et risus.

R

Maecenas semper, dui vitae ultricies elementum, neque nibh placerat mi, hendrerit gravida libero enim eget erat.

W

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Donec blandit interdum dolor ac laoreet.

Q

Etiam rutrum velit pharetra augue pellentesque facilisis. Praesent elit purus, cursus sodales molestie vel, fringilla vitae lectus.

X

Cras bibendum elit eu enim facilisis id feugiat orci interdum. Sed consequat fringilla consectetur.

A

Praesent eget lacus ut purus placerat pulvinar. Phasellus eget est nunc, eu pretium massa. Integer gravida nulla tempus eros dapibus ac ornare nunc mollis. Morbi vitae dolor id risus viverra feugiat.

H

Proin sapien risus, pulvinar eu volutpat eget, porttitor hendrerit ipsum.

B

Sed euismod molestie nibh a placerat. In condimentum blandit fermentum.

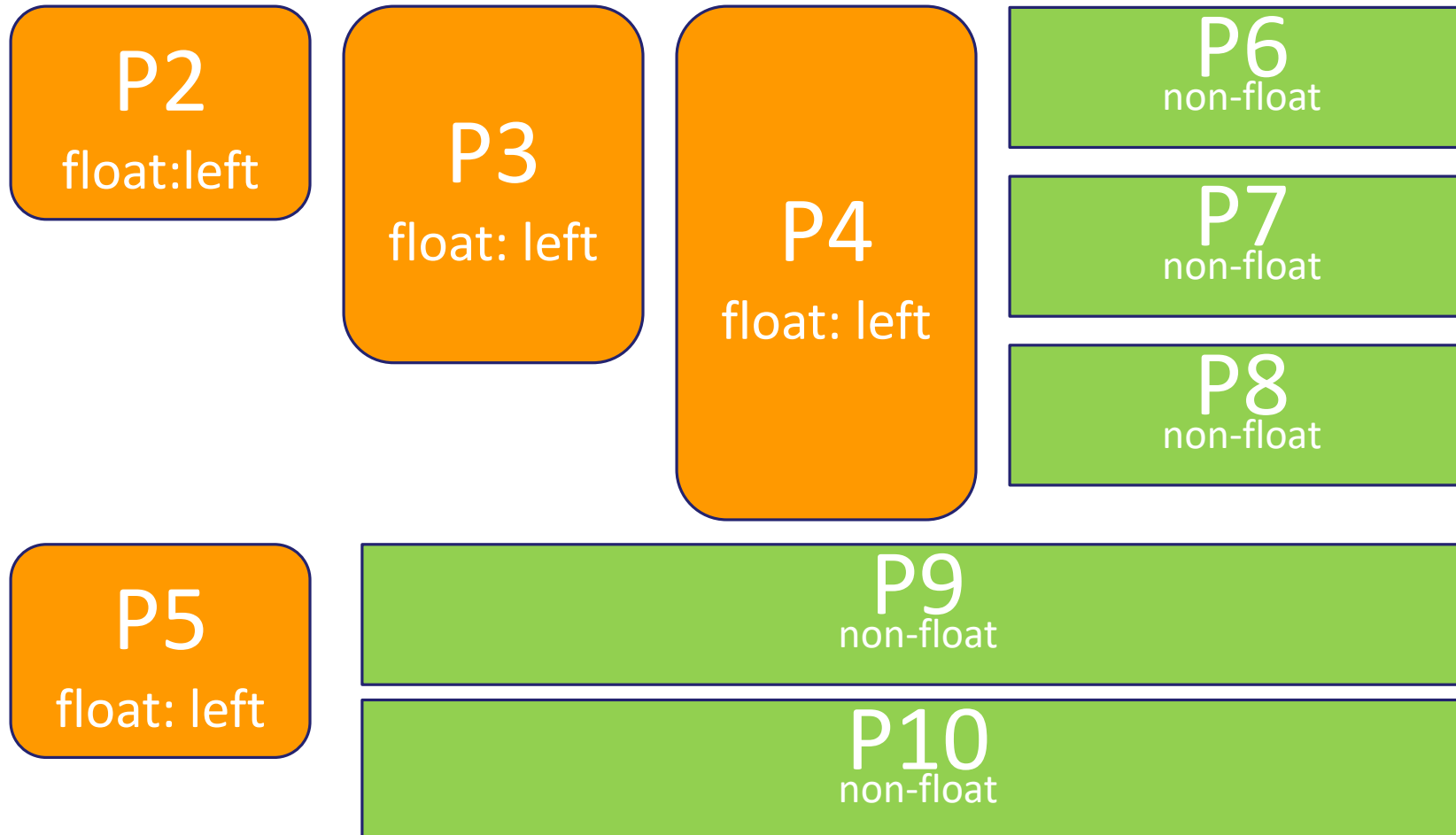
# CSS Float following the line to float to the left.

---



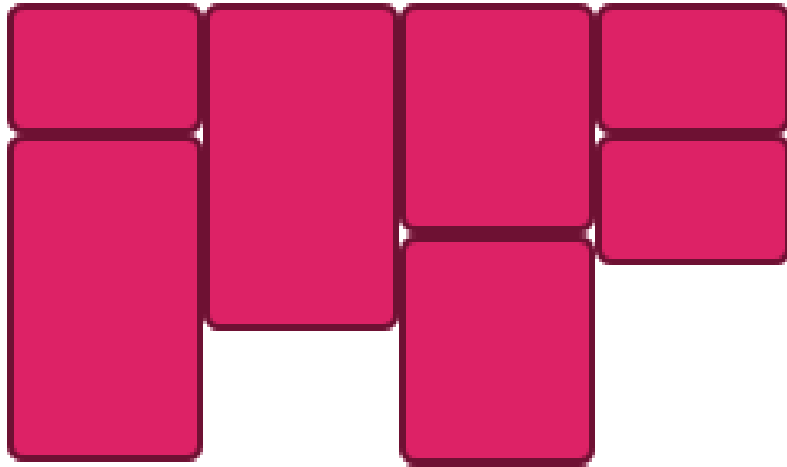
# Non-floats fill in the empty space if following the flow.

Study page 350-356

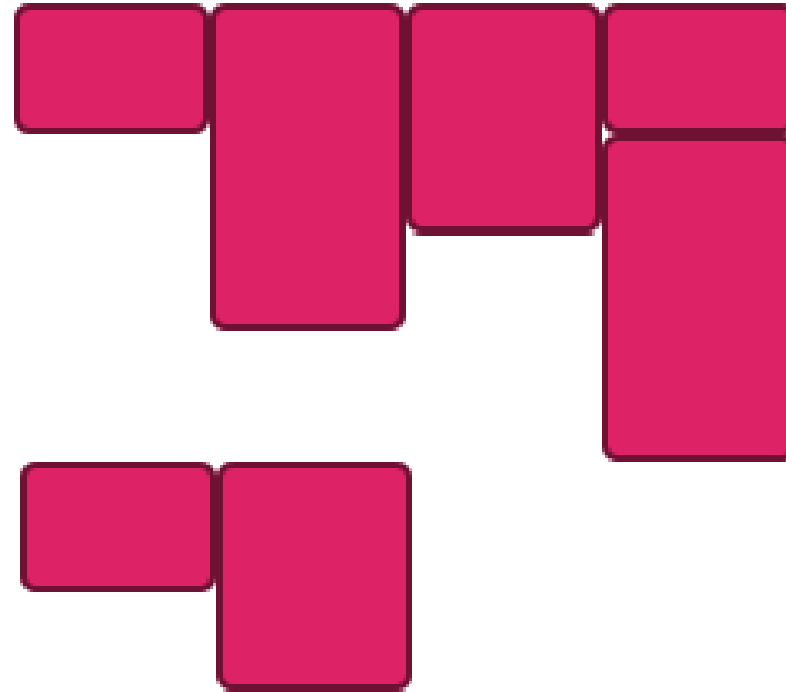


# What can still be improved for floating multiple elements

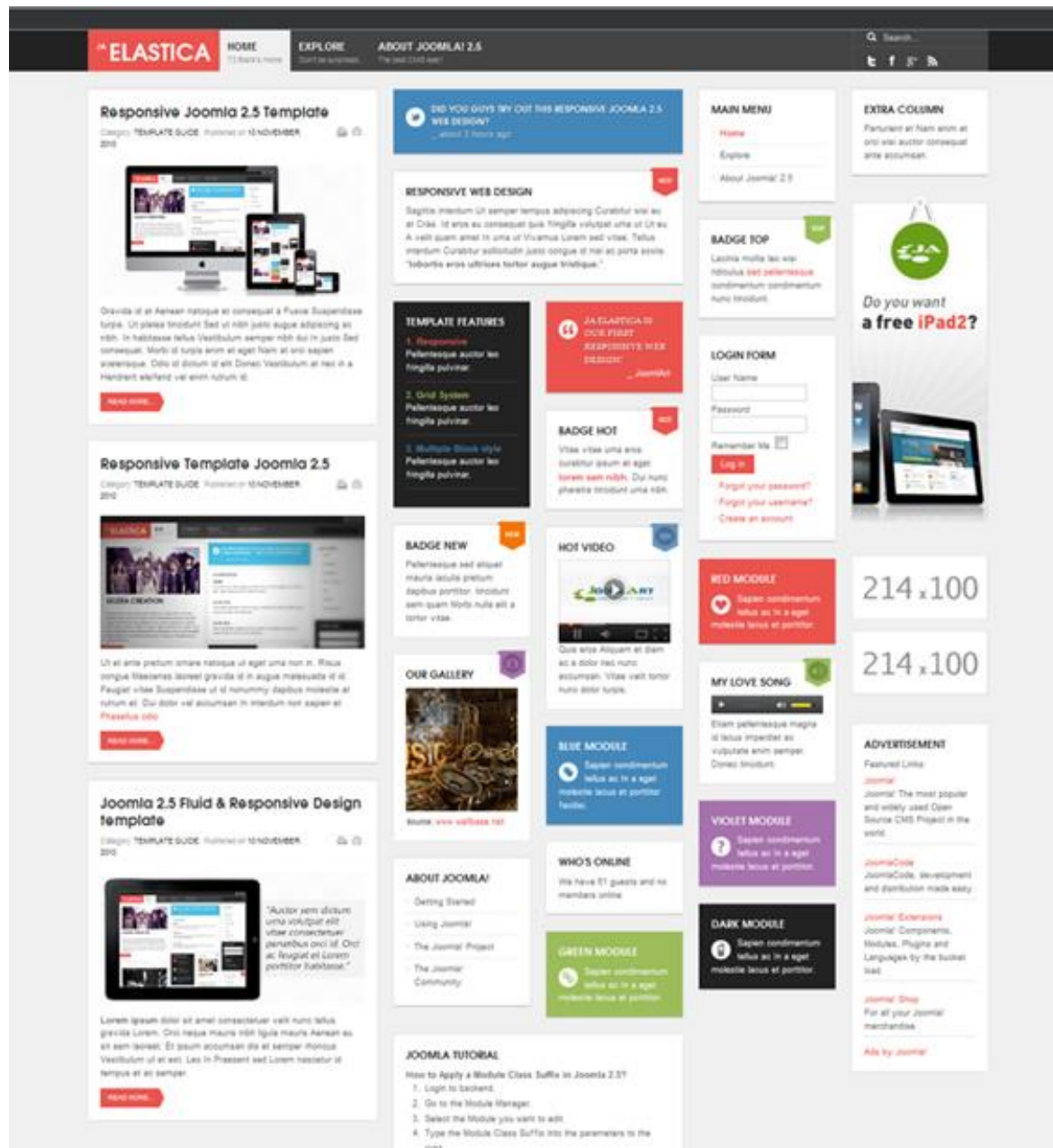
What you want



What you most likely get







# A Masonry Page



# Positioning

---

## LECTURE 3



# Positioning

position: static | relative | absolute | fixed | inherit

---

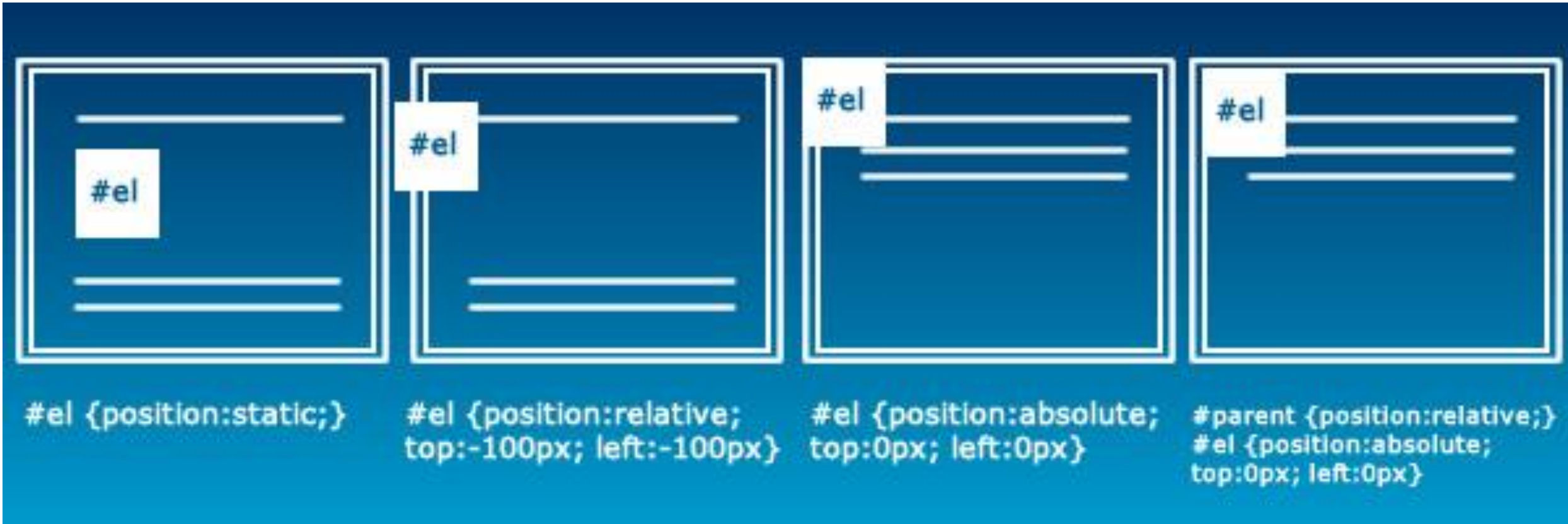
**static:** Positioned as they occur in the normal flow. (default)

**relative:** relative positioning moves the box **relative to its original position** in the flow.

**absolute:** Absolutely positioned elements are removed from the document flow entirely and positioned **with respect to the browser window or a containing element**.

**fixed:** The element **stays** in one position **in the window** even when the document scrolls.



static | relative | absolute to window | absolute to parent



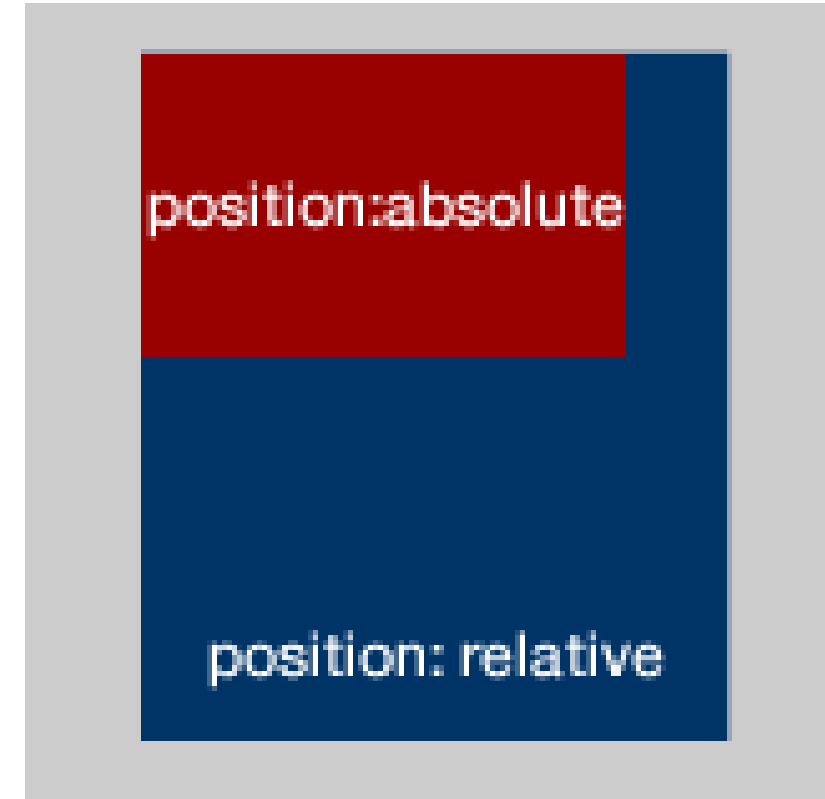
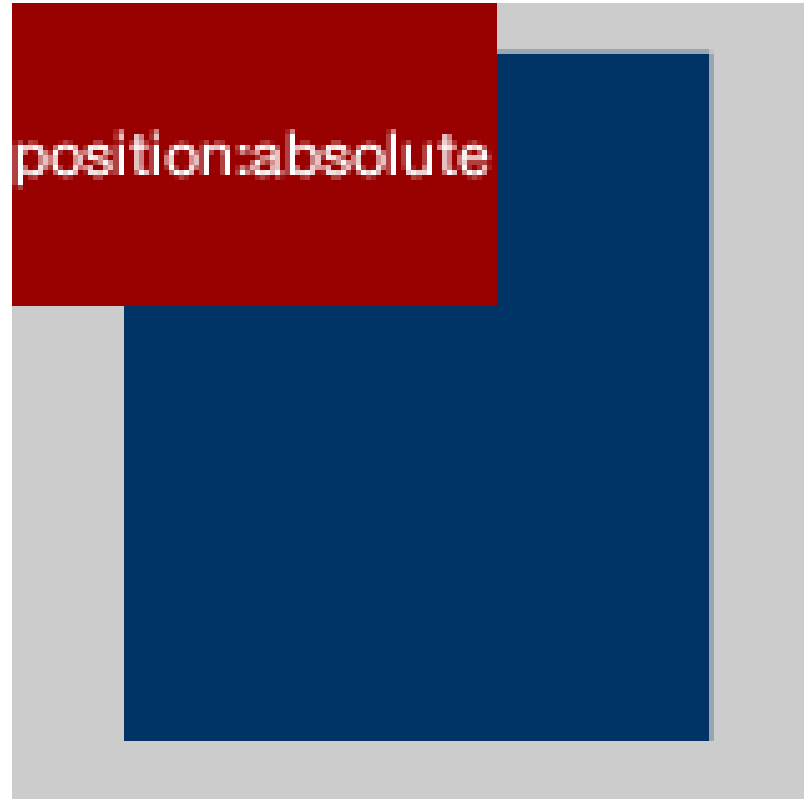
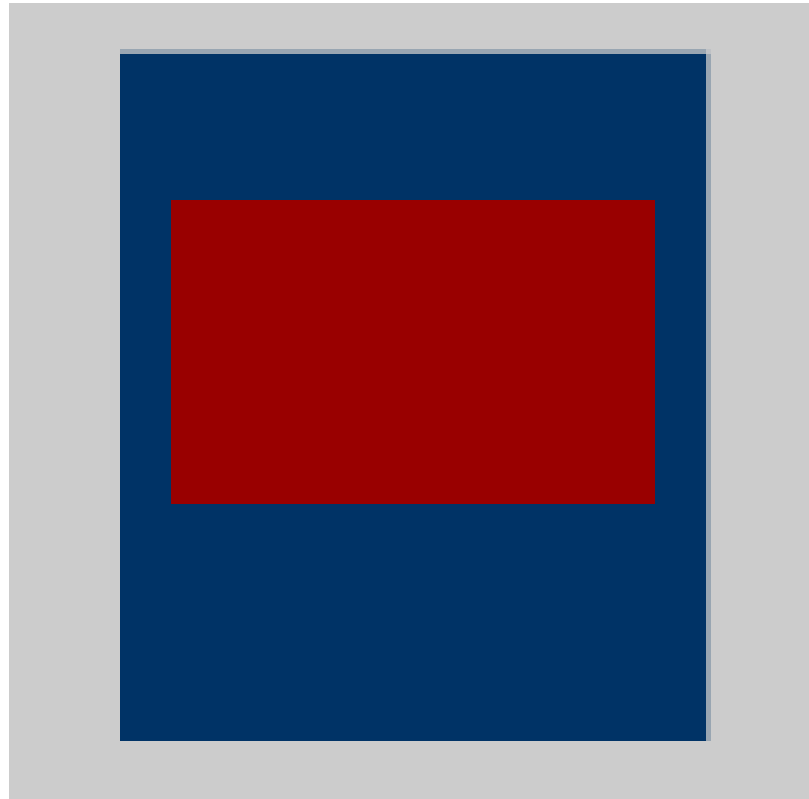
# Relative positioning

```
div.b{  
    position:relative;  
    top:20px;  
    left:20px;  
}
```



-  - new position of the element
-  - old position of the element

Another look at absolute to parent and absolute to window.



# Absolute to Window (no relative parent)

```
div.c{
  position: absolute;
  top:20px;
  left:20px;
}
```



# Absolute toe Parent (relative)

```
<div class="b">
  <div class="c"></div>
</div>
```

```
div.b{position: relative; }
div.c{
  position: absolute;
  top:20px;
  left:20px;
}
```





# top, right, bottom, left

top, right, bottom, left: length measurement | percentage | auto | inherit

---

- These settings take effects based on the position setting.
- Percentage positioning numbers are better (for responsive design)
- Read pages 357-366 and work on Ex 15-4,



# Stack Order and Z-index

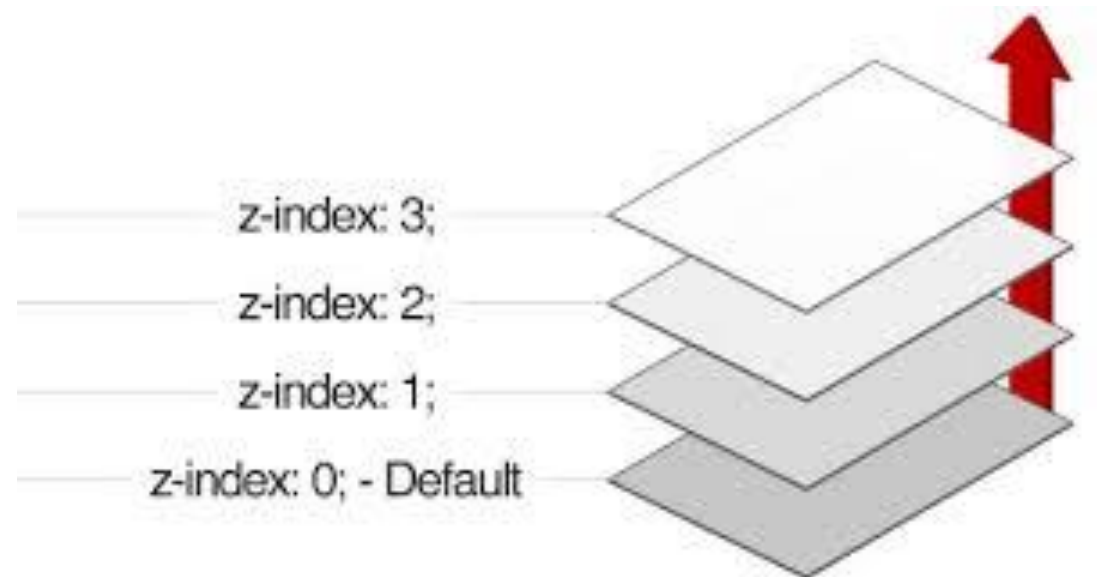
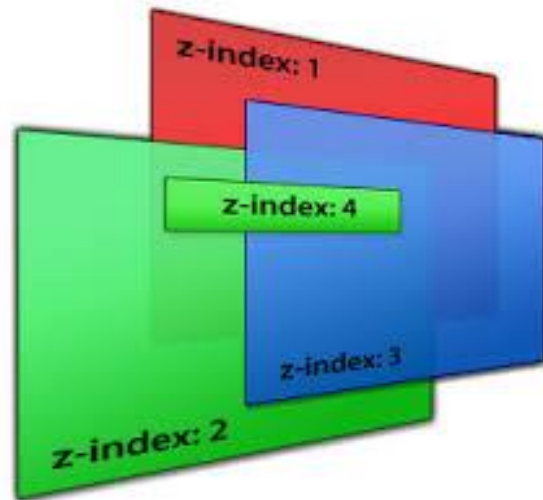
---

LECTURE 4

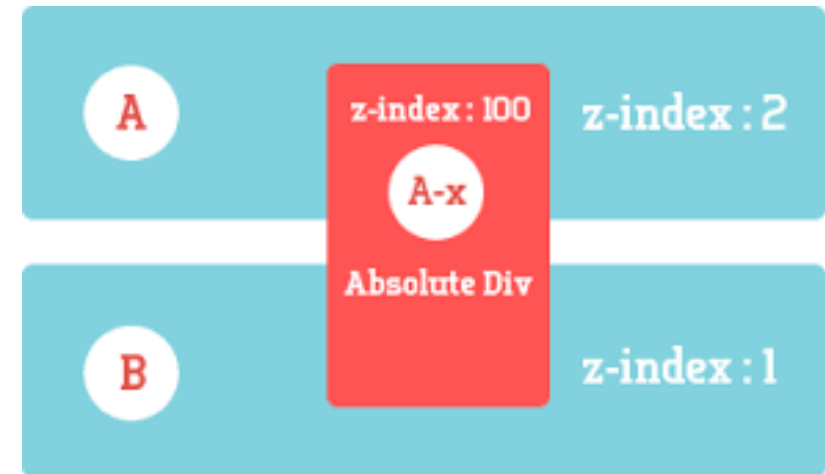
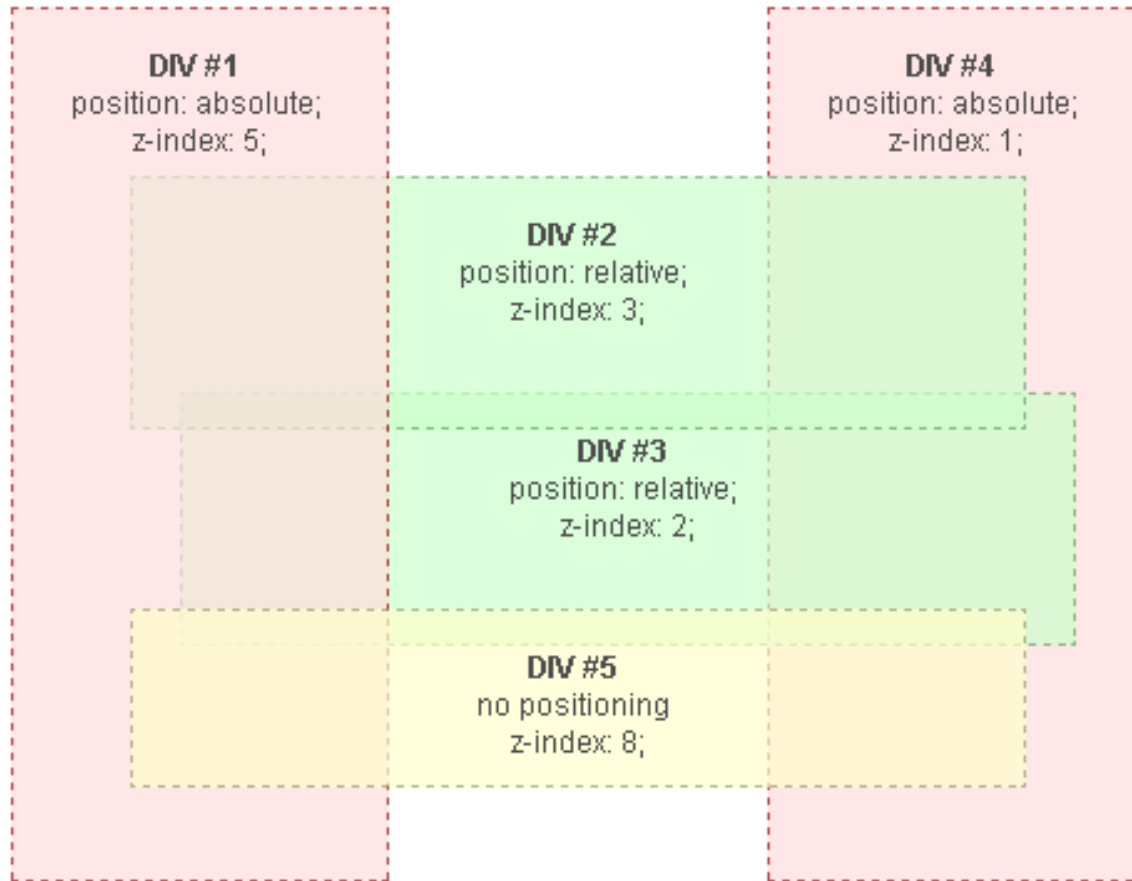
# Stacking order

z-index: number | auto | inherit

The value of the z-order property is a number (positive or negative). The higher the number the higher the element will appear in the stack. Lower numbers and negative values move the element lower in the stack.



# z-index provides solution when position overlay happens





# Masonry JavaScript Library

---

LECTURE 5

# Masonry JavaScript Library (optional)

---

- In this case it would be really helpful to have a horizontal version of the default CSS floating. Instead of only arranging elements horizontally you also want to float them vertically, positioning each element in the next open spot in the grid.
- And that's when the **Masonry JavaScript library** comes into play.
- **Masonry**, developed by David DeSandro, is a JavaScript library, that places your selected elements in an optimal position based on available vertical space.
- As its name implies, it fits your elements in your grid like mason stones in a wall. As a result, vertical gaps between elements of varying heights get minimized.

# Masonry JavaScript Library (optional)

---

- To get started with Masonry you only need to **add a single JavaScript file** to your Drupal site. There are several ways to do so.
- I prefer to either add the script within the head section of my HTML template file by using the script tag.
- You can also add the script path to the .info file of your theme. In the following example, we will also use **jQuery** which comes with Drupal by default. However please note that jQuery is not required to use Masonry.

# Masonry JavaScript Library (optional)

---

- Masonry works for every container element that has a group of similar child items. In the case of our Drupal view, the container element is the element inside the view with the class "view-content".
- The view rows with the class "views-row" are the child items. You can initialize the plugin either inline in your HTML by adding the class "js-masonry" to the container element as well as setting the data-masonry-options attribute or with a few lines of JavaScript code.
- I strongly recommend separating markup and functionality on a website. That's why I prefer the JavaScript option.



# Masonry JavaScript Library (optional)

---

- To initialize Masonry you create a new Masonry instance in one of your JavaScript files which is also included in the Drupal site. The Masonry constructor accepts two arguments: The container element and an options object.
- The container element can be selected with jQuery and then passed to the function, along with a variety of non-required options. However setting the **itemSelector** option is always recommended to make sure the correct child elements are specified as item elements.

# Masonry JavaScript Library (optional)

---

- After selecting the container element and setting the itemSelector option your JavaScript should look like this:

```
var container = $('.view-name.view-display-id-display_name .view-content');
```

```
var msnry = new Masonry( container, {  
    // options  
    itemSelector: '.views-row';  
});
```

# Masonry JavaScript Library (optional)

---

- You can minimize the lines of code by using **Masonry** as a jQuery plugin and applying it directly to the selected container element:

```
$('.view-name.view-display-id-display_name .view-  
content').masonry({  
    itemSelector: '.views-row'  
});
```

# Masonry JavaScript Library (optional)

---

- There are several options you can define inside the options object as e.g. the column width of your grid and the gutter between the columns. As long as no column width is set, all sizing of your grid is handled by your CSS. The Masonry plugin acts responsively. It adjusts its positioning to the column width set in your CSS mediaqueries. If the options are not sufficient, there are also actions you can take on your masonry instance dynamically. Those methods allow you e.g. to append or prepend further elements to your grid, remove certain elements and redefine the layout of your grid in general.
- After studying all those methods it appears you can most anything you'd need to with Masonry. So next time you're working with grid layouts, surf over to <http://masonry.desandro.com/> and give it a shot.



# jQuery Masonry

---

LECTURE 6

# jQuery Masonry

---

- jQuery Masonry is the most popular solution to pull off this type of layout. It utilizes some pretty fancy JavaScript to reflow a series of divs.
- Putting Masonry into practice is pretty easy, all you need is a container that holds a series of divs that you want to arrange masonry style. You can place anything you want inside the divs, in this case I threw in some placeholder images.
- Once you have that in order, toss in jQuery and jQuery Masonry. Then you need to create a simple function that identifies your container and targets the class that we used for our Masonry images divs.



# HTML

---

```
<div id="container">
  <div class="masonryImage">
    
  </div>
  <div class="masonryImage">
    
  </div>
  <div class="masonryImage">
    
  </div>
  ...
</div>
```



# JavaScript

---

```
<script src="jquery-1.7.1.min.js"></script>
<script src="jquery.masonry.min.js"></script>
<script>
  $(function(){
    var $container = $('#container');
    $container.imagesLoaded( function(){
      $container.masonry({
        itemSelector : '.masonryImage'
      });
    });
  });
</script>
```





# CSS

---

```
#container {  
  width: 1200px;  
  margin: 0 auto;  
}  
/*Media Queries*/  
@media only screen and (max-width : 1199px),  
only screen and (max-device-width : 1199px){  
  #container {  
    width: 1000px;  
  }  
}  
@media only screen and (max-width : 999px),  
only screen and (max-device-width : 999px){  
  #container {  
    width: 800px;  
  }  
}
```



# CSS

---

```
@media only screen and (max-width : 799px),
only screen and (max-device-width : 799px){
  #container {
    width: 600px;
  }
}
@media only screen and (max-width : 599px),
only screen and (max-device-width : 599px){
  #container {
    width: 400px;
  }
}
@media only screen and (max-width : 399px),
only screen and (max-device-width : 399px){
  #container {
    width: 200px;
  }
}
```

# Example

---



# See It In Action

---

- This gives us a nice tight image gallery that perfectly responds to various browser window sizes. I've taken all the spacing out to illustrate just how tight you can get your images, but feel free to build in some margins so your images are spaced out a bit more.

<http://designshack.net/tutorialexamples/masonry/demos/masonry.html>