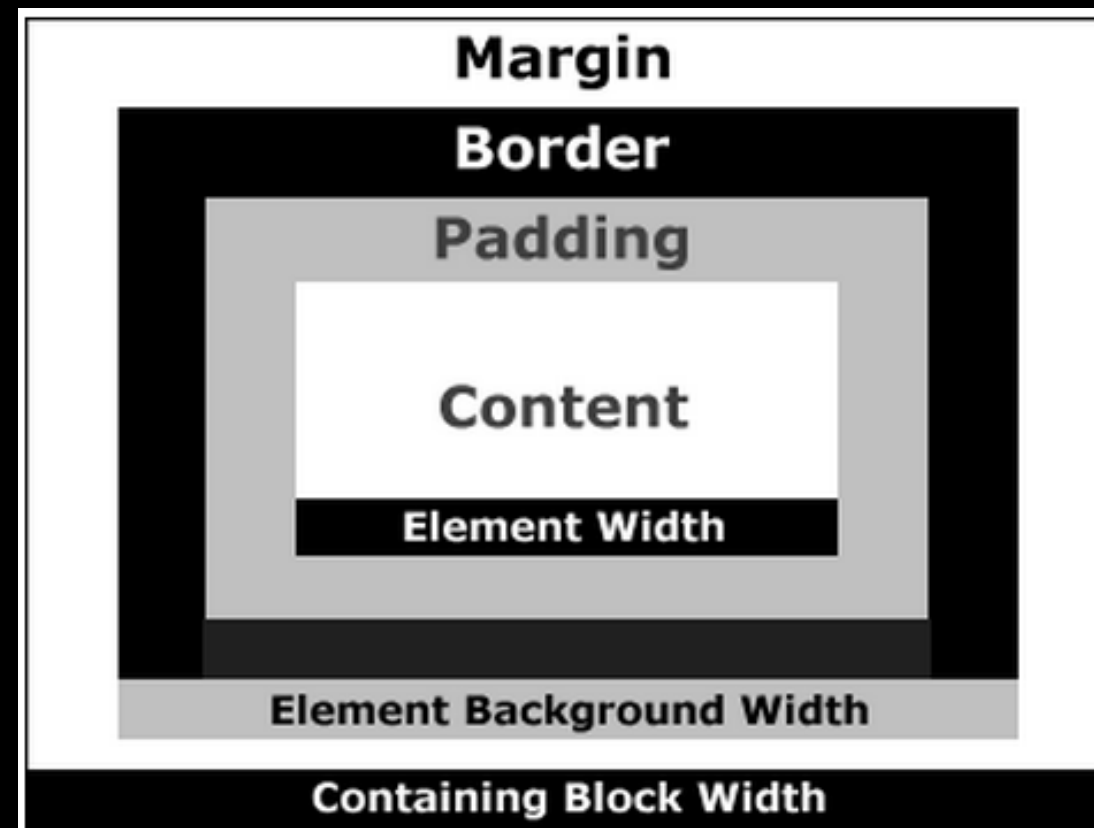


CSS Layout

Examining page layout methods with CSS

The Box Model



In order to layout pages properly with modern CSS, you'll need to understand the box model

The Box Model

- ☞ The content is the text inside of an HTML element
- ☞ The padding is space within an element's border or inside of margin if there is no border
- ☞ The border forms a line around the element
- ☞ The margin is space outside of the element's border or outside of the padding if there is no border

Margin and Padding

```
margin: 20px;
```

Put a 20 pixel margin around the element or on the left and right side if the element is inline

```
padding: 50px;
```

Put 50 pixels of padding around the element or on the left and right side if the element is inline

Margin and Padding

```
padding-right: 10px; margin-left: 40px;
```

Append -right, -left, -top, -bottom to the padding or margin commands to be more specific

```
padding: 20px 10px; margin: 100px 30px;
```

The first value is padding or margin on the top and bottom, the second is padding or margin on the left and right

Centering content: a trick

```
width: 900px;  
margin: 0 auto;
```

- ☞ Setting the left and right margin to `auto` while using a fixed width centers an element horizontally
- ☞ Generally, these attributes are set on a `div` with the id `wrapper`
- ☞ This `div` is wrapped around a site's content so it is centered, even when the window is resized

Border Styles

```
border-style: dashed;
```

Gives the border a dashed style. Other possible values include dotted, solid, double, groove, ridge, inset, and outset

```
border-width: 10px;
```

The border should be 10px wide

Border Styles

```
border-color: #111111;
```

Gives the border a gray-ish color specified using a hex code

```
border: 1px solid red;
```

Puts a 1 pixel solid red border around an element, shorthand for all of the attributes gone over so far

Exercise #1: Using the box model

Create a `<p>` element with some text in it and try giving it `padding`, `border`, and `margin` to see the effects each of these properties have on the element

inline vs block

☞ Inline elements can't have margin/ padding on their top and bottom & can be next to other elements

<!-- Elements that are normally inline -->

<i></i>

☞ Block elements can have margin/padding on their top or bottom & cannot be next to other elements

<!-- Elements that are normally block -->

<p></p>

<div></div>

You can also change an element's default behavior using the `display` property.

```
display: block;
```

The element is displayed as a block, just like a `<p>` tag is, tolerates no elements aligned next to it unless `float` is used

```
display: inline;
```

The element is displayed inline, like a `` tag, inside the current block on the same line as other elements it is near

```
display: inline-block;
```

A combination of the two which allows a block to be next to another block, but still have vertical padding and margins

Exercise #2: `inline` vs `block` vs `inline-block`

- ☞ Create 4 `<p>` elements with 4-5 sentences of `lorem ipsum` inside of them, with some of these words selected with `` tags
- ☞ Give all of the elements different classes, give them a set width (say 250px) and try seeing what happens when you change `block` elements to `inline` elements and vice-versa
- ☞ Also experiment with the `inline-block` tag
- ☞ To more clearly see what's going on, try putting a border around your elements

float

```
float: left;
```

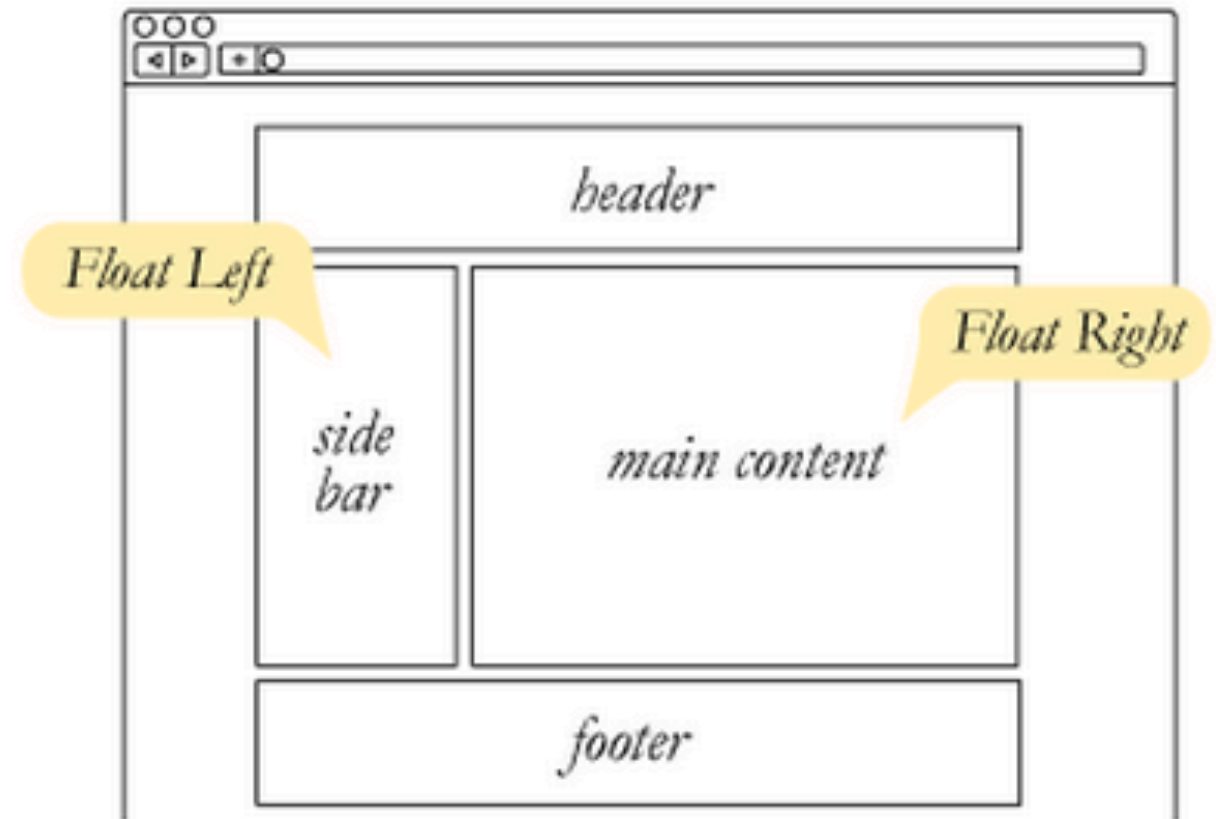
Move an element all the way to the left side of its container. Have all other elements flow around it.

```
float: right;
```

Move an element all the way to the right side of its container. Have all other elements flow around it.

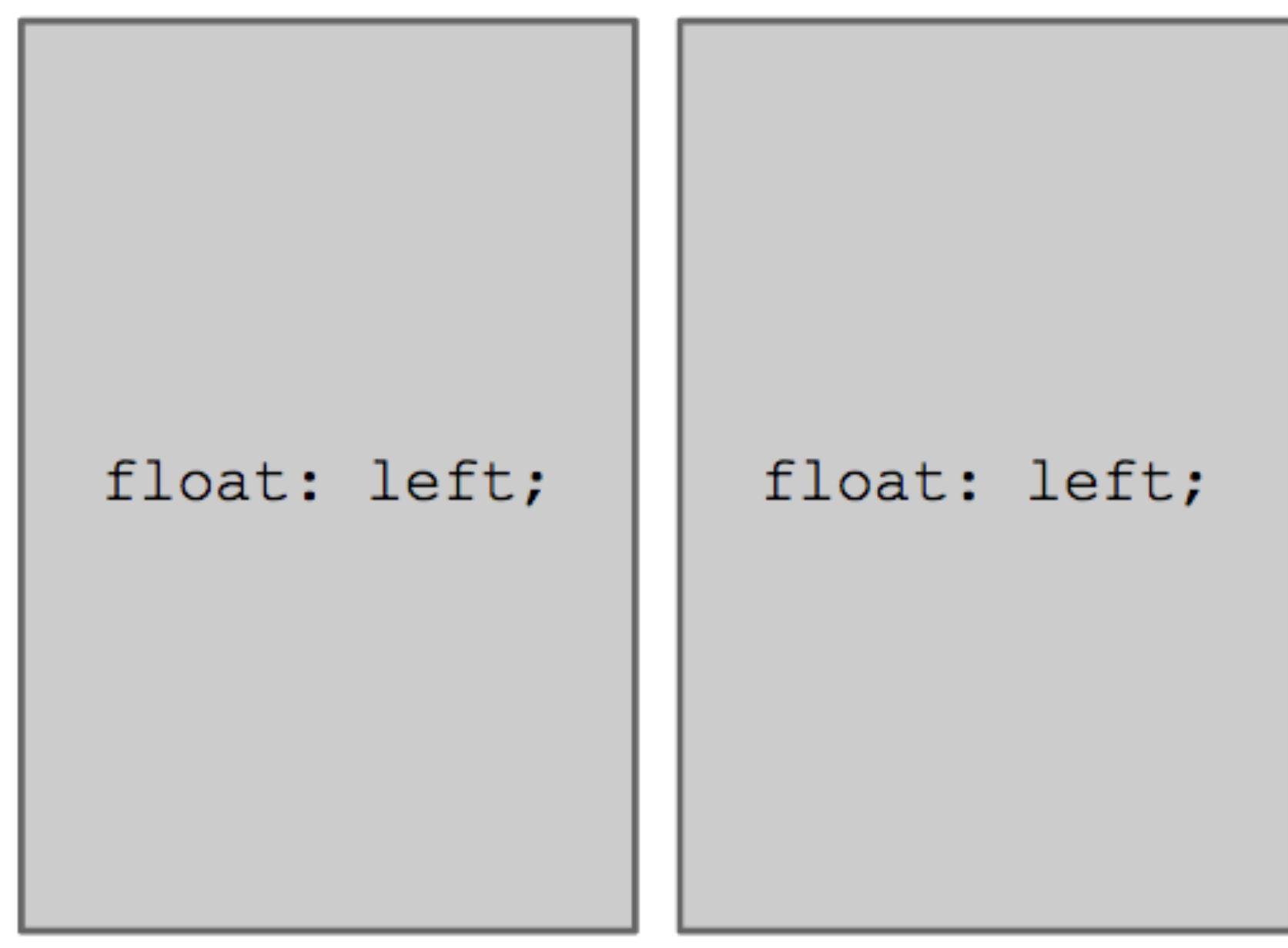
```
clear: both;
```

Remove the effects of a floated container, applied to an element after a floated container to clear the effects of using float



Using **float** for page layout

- ☞ To create a two-column page layout, float the first and second column left
- ☞ Try resizing the page to a smaller width - the second column will collapse under the first (left) column

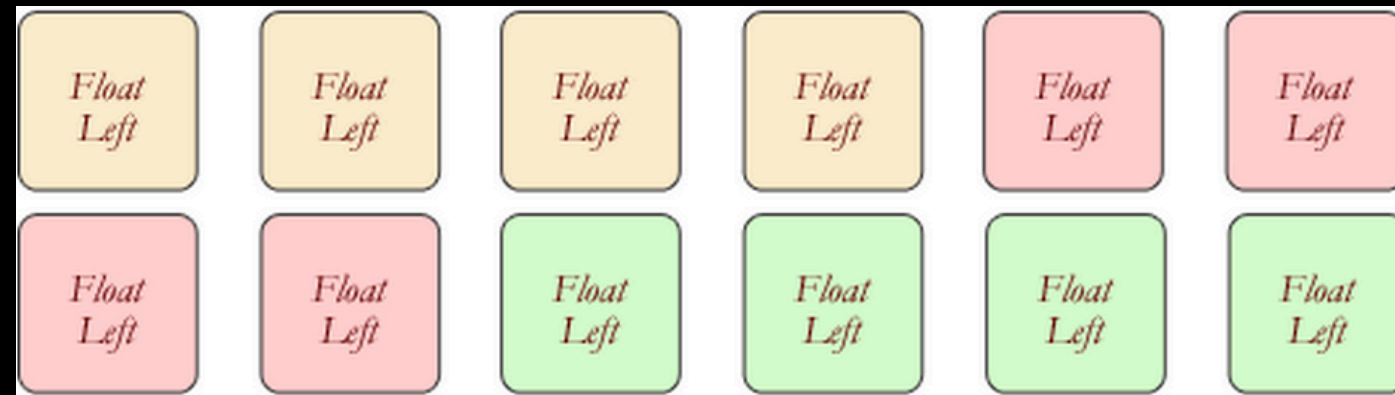
The diagram illustrates a two-column layout using the CSS float property. It consists of two adjacent gray rectangular boxes, each representing a column. Inside each box, the text 'float: left;' is written in a monospaced font. The boxes are separated by a thin white vertical line, and the entire pair is enclosed within a white border.

```
float: left;
```

```
float: left;
```

Using `float` for page layout

- ☞ You could use `float: left;` to create a dynamic image gallery
- ☞ `float` each image left and give it a `margin-right` and `margin-bottom` to enforce space



- ☞ As you resize the page, the images will stack nicely



Using `float` for page layout

- ☞ To add an image to the page with text flowing around it, float it left
- ☞ Try adding some `margin-right` and `margin-bottom` to the image

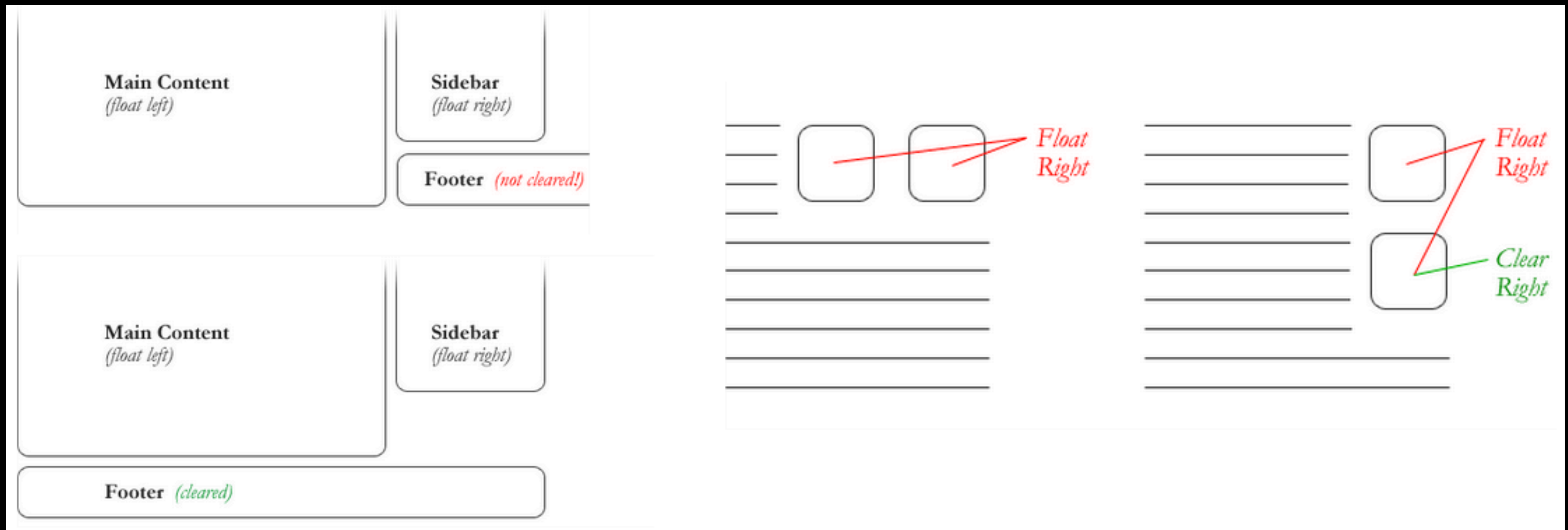
Exercise #3: floats

- ☞ Create 2 `<p>` elements with 4-5 sentences of `lorem ipsum` inside of them
- ☞ Also put an `` element on the page above the `<p>` elements
- ☞ Give all of the elements different classes and try floating the `` first, than the `<p>` elements to get an idea of how floats impact page layout

More on `clear`

- ☞ The `clear` property will move the element down past surrounding floated elements
- ☞ `clear` has 4 possible values: `both`, `left`, `right` and `none` (default)

More on clear



Positioning with CSS

- ☞ When the usual tricks (inline vs block, floats, etc) aren't working to position your elements on the screen, try using CSS positioning
- ☞ Positioning should be viewed as a last resort in most cases

Positioning with CSS

- ✎ Using the CSS position attribute, you can use X and Y values to move elements around the screen based on different frames of reference
- ✎ There are four possible values for the CSS position:

```
static /*This is the default value*/  
fixed  
absolute  
relative
```

left, right, top, and bottom attributes

- ✎ left, right, top, and bottom are used to specify an offset, the reference point of which is determined by the type of positioning specified (fixed, absolute, relative, etc.)
- ✎ For left, offset values that are positive will move your element to the right while negative values will move it to the left.
- ✎ right does the opposite from the other side of the screen
- ✎ For top, offset values that are positive will move your element down while negative values will move it up
- ✎ bottom does the opposite from the bottom of the screen

relative

- ☞ Using `position: relative;` you can position your element relative to where it would normally be on screen
- ☞ The element is still in the normal "flow" of the page, it still takes up space
- ☞ Works well to make small changes in position unachievable through margin or padding

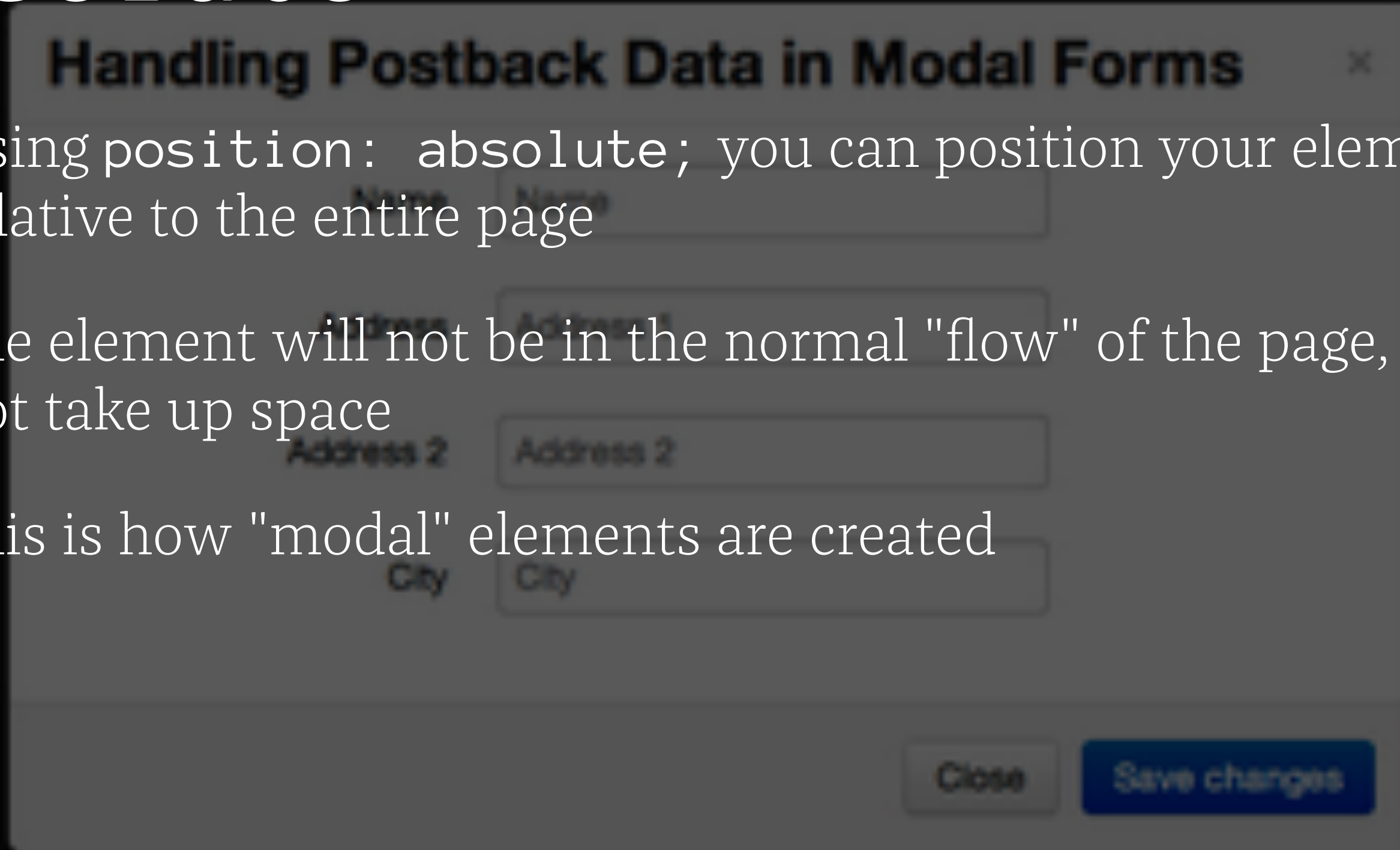
Positioning Example

```
<div class='move-me'>  
  Some content  
</div>
```

```
/* Move the words 'Some content' 10 pixels down  
from where it would normally be*/  
.move-me{  
  position: relative;  
  top: 10px  
}
```


absolute

- ➡ Using `position: absolute`; you can position your element relative to the entire page
- ➡ The element will not be in the normal "flow" of the page, it will not take up space
- ➡ This is how "modal" elements are created



Final Exercise

Create a fake website for a newspaper, "The New York Code + Design Academy Times"

- ☞ There should be two pages:
 - ☞ Home page, where 10 fake articles are listed in a two-column layout - they should all link to:
 - ☞ An example article with a link to Facebook that stays on the page no matter how much the page is scrolled
 - ☞ The example article should have a photo with text that wraps around the photo (hint: use a `float`!)
- ☞ Use the `margin: 0 auto; width: 900px;` "trick" to make the pages look nice and centered
- ☞ If you finish the above, have fun with CSS making the newspaper look as professional as possible!
- ☞ Don't worry about any of the actual text - just use lorem ipsum