CSS, Part 3: Positioning

In this slide show...

The Box Model: padding, margins, borders, drop shadows, rounded corners, and sizing

Positioning Properties: floating, clearing, relative positioning, absolute positioning, fixed positioning, overflow, and display roles

Page Layouts: fixed, fluid, and elastic layouts

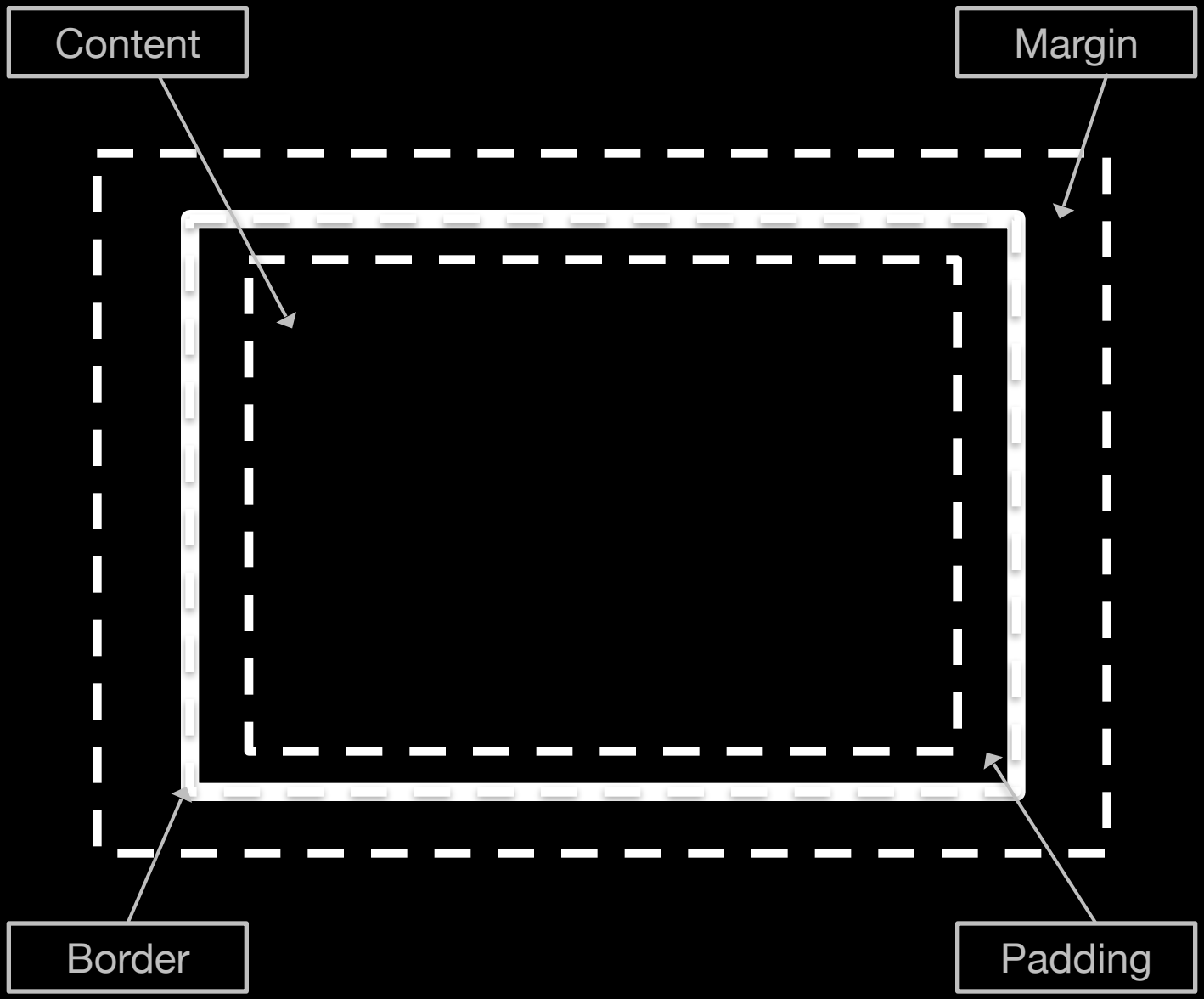
The Box Model

CSS box model

Think of every element in your HTML as having an imaginary box around it

You can control: the space inside the box before the content starts (padding); the space outside the box (margin); and the line around the box (border)

By combining these three, you can do a ton of layout work without really digging into positioning

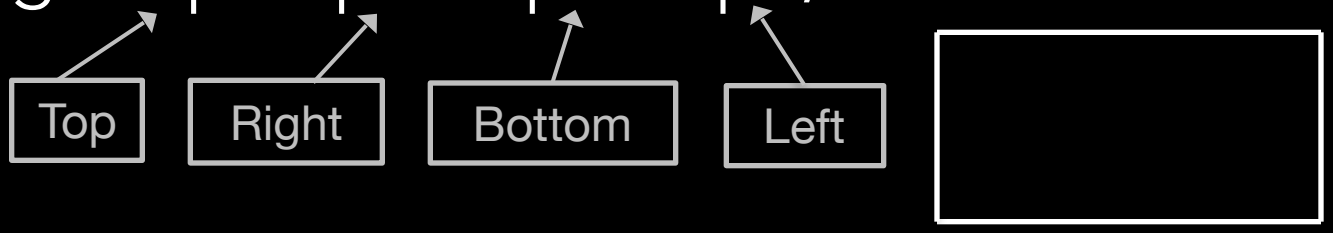

Content
Margin
Border
Padding

Syntax for padding and margin

padding: and margin: properties

Add -top, -bottom, -left, or -right to specify particular edge (e.g., padding-top)

padding: 10px 5px 15px 20px;



padding: 10px 5px; Top and bottom, then left and right.

padding: 10px; All four sides at once.

Padding

Space inside the box

Allows you to give your content more room without increasing space between content boxes

Valid values include all length measurements and percentages

Default value: 0px

Margins

Space outside the box

Allows you to add space between items (e.g., at the end of a paragraph)

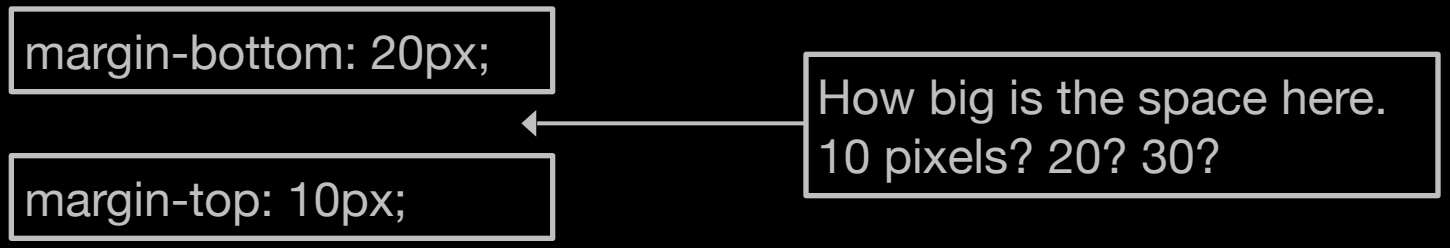
margin: 0 auto; automatically calculates left and right margins, is used to center content

Valid values include all length measurements and percentages

Default value: 0px

Margins (cont.)

Margins naturally collapse, meaning that two overlapping margins will default to the larger value



Negative margins are valid values

On *inline* content, left and right margins display, top and bottom do not

Syntax for borders

Three different components of a border:

- border-width, border-style, and border-color

Order is the same as padding and margin

border: property combines all three together; only use when all four sides are same

border: 1px solid black;

Values for border-style

none

dotted

dashed

solid

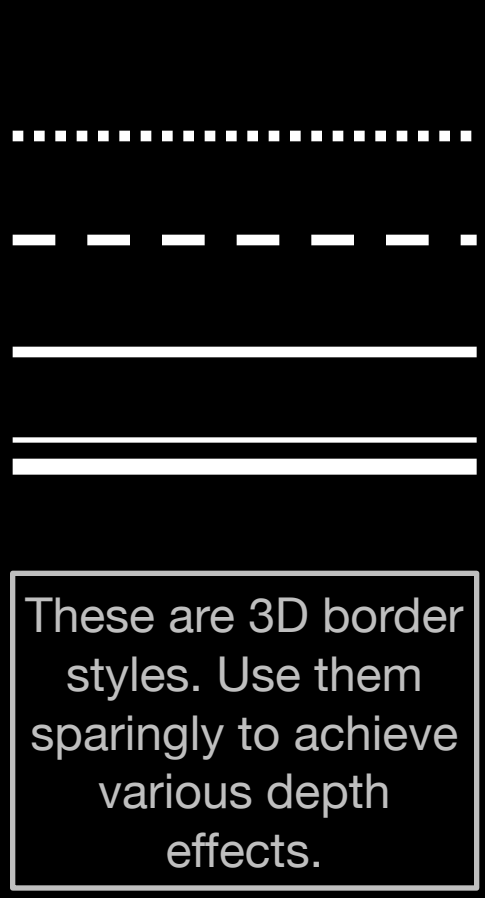
double

groove

ridge

inset

outset



These are 3D border styles. Use them sparingly to achieve various depth effects.

How to use borders

To divide parts of the page

To create lines between content

To delineate tables

To underline links

See me if you want to learn how to use an image for a border with border-image: property

Sizing box elements

All element boxes are automatically sized

You can specify size, either in length measurement (for fixed size) or percentage (for variable size)

Use width: and height: properties

In general, you will want to specify width, but not height

Rounded corners

- border-radius: property easily creates rounded corners
- Syntax is the same as border-width:
- Valid values include length measurements and percentages
- New to CSS3: works in all current browsers, but not in many older browsers

Drop shadows

box-shadow: property easily creates drop shadows on boxes

Specify horizontal value first, then vertical value

Can (but not required to) add blur, size, and color of shadow

Add inset at the end to put the shadow on the inside of the box

Can use a box-shadow generator online

box-shadow: 3px 3px 2px rgba(0,0,0,0.4);

Positioning Properties

Positioning items with CSS

In general, there are three positioning properties you need to know: float, clear, and position

float: allows you to position several items side-by-side; used for creating columns and wrapping text

clear: helps reset the document to normal after float:

position: allows you to fine tune position

Floating elements

float: property pushes a block element as far as it can go to the left or right and wraps following items around it

Using float: removes the element from the normal flow of the document

Inline floated elements become block

float: can be applied to images to wrap text around them

float: can also be applied to content areas to create columns

Floating images

```
img { float: left; padding-right: 20px; }
```

Here's the text that wraps directly around the image. You don't have to do anything to the paragraph to have it wrap. All you have to do is float the image.

Floating to create columns

In your HTML, create content dividers (elements with classes/IDs or HTML5 semantic elements)

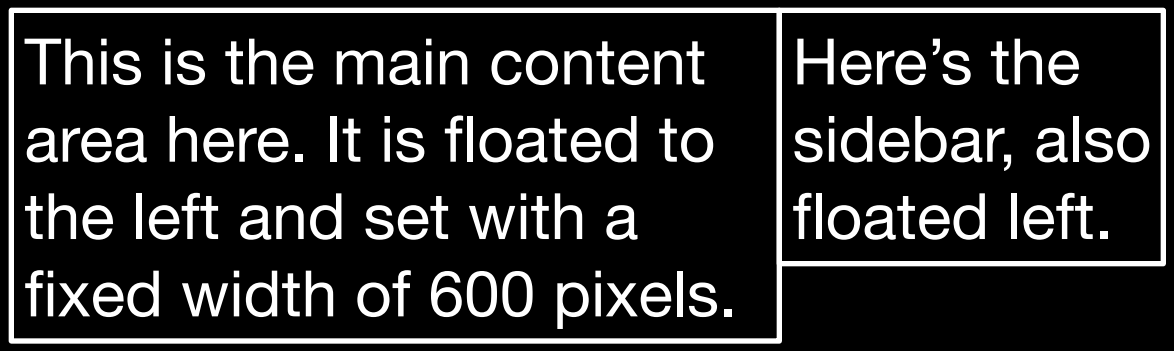
Arrange your content dividers in the vertical order you want them to appear on the page, left to right

Float content dividers left

Theoretically, you can create as many columns as you want this way

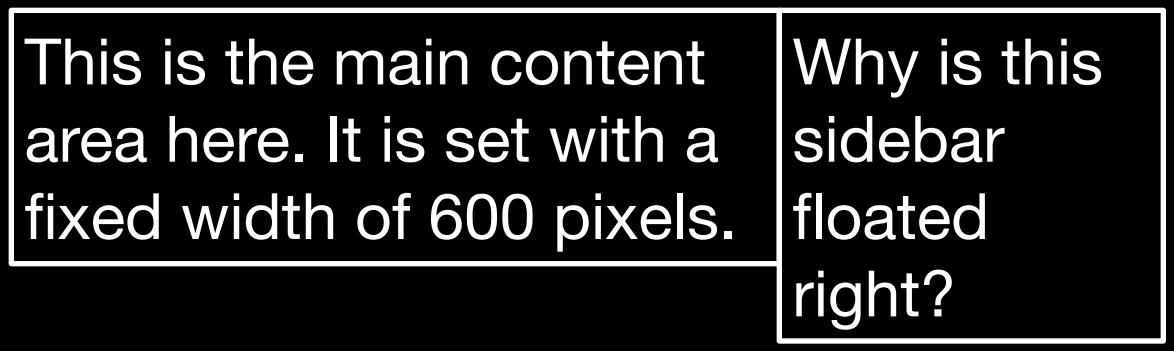
Floating to create columns (cont.)

```
<div id="main"></div>
<div id="sidebar"></div>
#main { width: 600px; float: left; }
#sidebar { width: 300px; float: left; }
```



Floating to create columns (cont.)

```
<div id="sidebar"></div>
<div id="main"></div>
#main { width: 600px; }
#sidebar { width: 300px; float: right; }
```



Floating to create columns (cont.)

```
<div id="leftsidebar"></div>
<div id="main"></div>
<div id="rightsidebar"></div>
#leftsidebar, #rightsidebar { width: 200px;
        float: left; }
#main { width: 600px; float: left; }
```

Can you tell me why I can get away with grouping the selectors for the left and right sidebars?

Clearing items

Because float: removes elements from the normal document flow, you will occasionally encounter hiccups

clear: property allows you to fix those hiccups

clear: left; pushes the item down until there is nothing to its left

clear: right; and clear: both; are also valid

Commonly used for footers

Clearing items (cont.)

```
<div id="main"></div>
<div id="sidebar"></div>
<div id="footer"> </div>
#main { width: 600px; float: left; }
#sidebar { width: 300px; float: left; }
#footer { clear: both; }
```

This declaration ensures that the footer does not run into the content area or sidebar.

Positioning with position

position: property allows you to fine tune positioning more

Three general types of positions: relative, absolute, and fixed

To use position: property, first create a rule with position: _____; as the declaration

Then add declarations with positioning measurements using left:, right:, top:, and bottom:

Relative positioning

position: relative;

Repositions the item relative to its current position inside its containing element

Does not remove item from document flow

Think of relative positioning as simply bumping an item up, down, or over

Absolute positioning

position: absolute;

Removes the item from the document flow

Positions the item at a set point in the containing block, with the measurement values as offsets

right: 0px; bottom: 0px; would position the item in the bottom-right corner of its containing block

Fixed positioning

position: fixed;

Acts the same as absolute positioning, except fixed position items are anchored, meaning they will not scroll with the rest of the content

Fixed positioning can create some hiccups, so use it sparingly

Use media queries to change to position: static; for print

Overlapping items and priority

When you use the position: property, you sometimes end up with overlapping elements

Use the z-index: property with any integer value (including negatives) to determine stack order

Items with higher z-index values display in front of items with lower values (e.g., 10 would display above 9)

Best practices in position: property

Use the position: property sparingly; use margins and padding to bump the content or a box in one direction

Use position: fixed; to create elements that don't scroll (e.g., static banner, sidebar, etc.)

For advanced layouts, combining positioning properties may be necessary

Overflow

overflow: property used to specify how the content that doesn't fit in a sized box (i.e., the overflow) displays

Valid values include: visible, hidden, scroll, and auto

Display roles

display: property allows you to change how an element displays

Valid values include: none, inline, block, and inline-block

Generally used only in specific scenarios

display: inline; used to make <li> in menu nav display horizontally

display: none; used to make content disappear from the document flow

Page Layouts

Page layouts

Three types: fixed, fluid, and elastic

Generally set widths, not heights

Fixed: specific pixel values, elements do not resize

Fluid: % values, elements resize with browser window

Elastic: em values, elements resize with font size (*very* hard to do well)

Fixed layouts

Allow for much more control over design

Are simpler and easier to code

Are problematic for small screens and large screens

Fluid layouts

Change based on device size, ensuring consistent proportions

Work on all devices

Are harder to code and test

Why not height?

Elements will naturally be as tall as the content requires them to be

Setting any height is almost certain to cut off your content

Your content needs somewhere to go when text or the browser window is resized

Problem: no height means unequal column heights

Solution: various workarounds (come see me)

Container

```
<div id="container">

</div>
```


```
#container {
        margin: 0 auto;
        width: ————————-;
        }
```

Adding columns to layout

```
<div id="container">
        <div id="banner"></div>
        <div id="nav"></div>
        <div id="sidebar"></div>
        <div id="main"></div>
        <div id="footer"></div>
</div>
```

As usual, these can be replaced by HTML5 semantic elements. The container should be a , though.

Floating columns to finish layout

```
#sidebar {
        float: left;
        width: ————;
        }
#main {
        float: left;
        width: ————;
        }
```

This float system requires you to order the elements correctly in HTML. Instead, if you float one column left and the other right, that will create the same layout, no matter which comes first in the HTML document.

Working with fixed widths

- Butter zone that will accommodate most users (760 - 960px)
- Make sure you subtract horizontal borders, padding, and margins when calculating widths
- Follow the rule of thirds: for 900px, 300+300+300

Working with fluid widths

Set minimum (min-width) and/or maximum (max-width) to ensure elements don't end up too small/big

Make sure you subtract horizontal borders, padding, and margins when calculating widths

If using lots of columns, use media queries to change column layout for smaller devices

Thanks for watching!