# HACK YALE

## < INTRO TO WEB DEVELOPMENT />

WWW.HACKYALE.COM

HACK YALE

< INTRO TO WEB DEVELOPMENT />

# Week 3

## MORE CSS

# SPACING, BORDERS, AND POSITIONING

HACK YALE

# Width and Height

The width and height properties can be used to specify the dimensions of an HTML element. They are *usually* used with `div`.

Values can be given in units of length (such as pixels) and in terms of a percentage of the containing element.

```
div.inner {
   width: 50%;
}
```

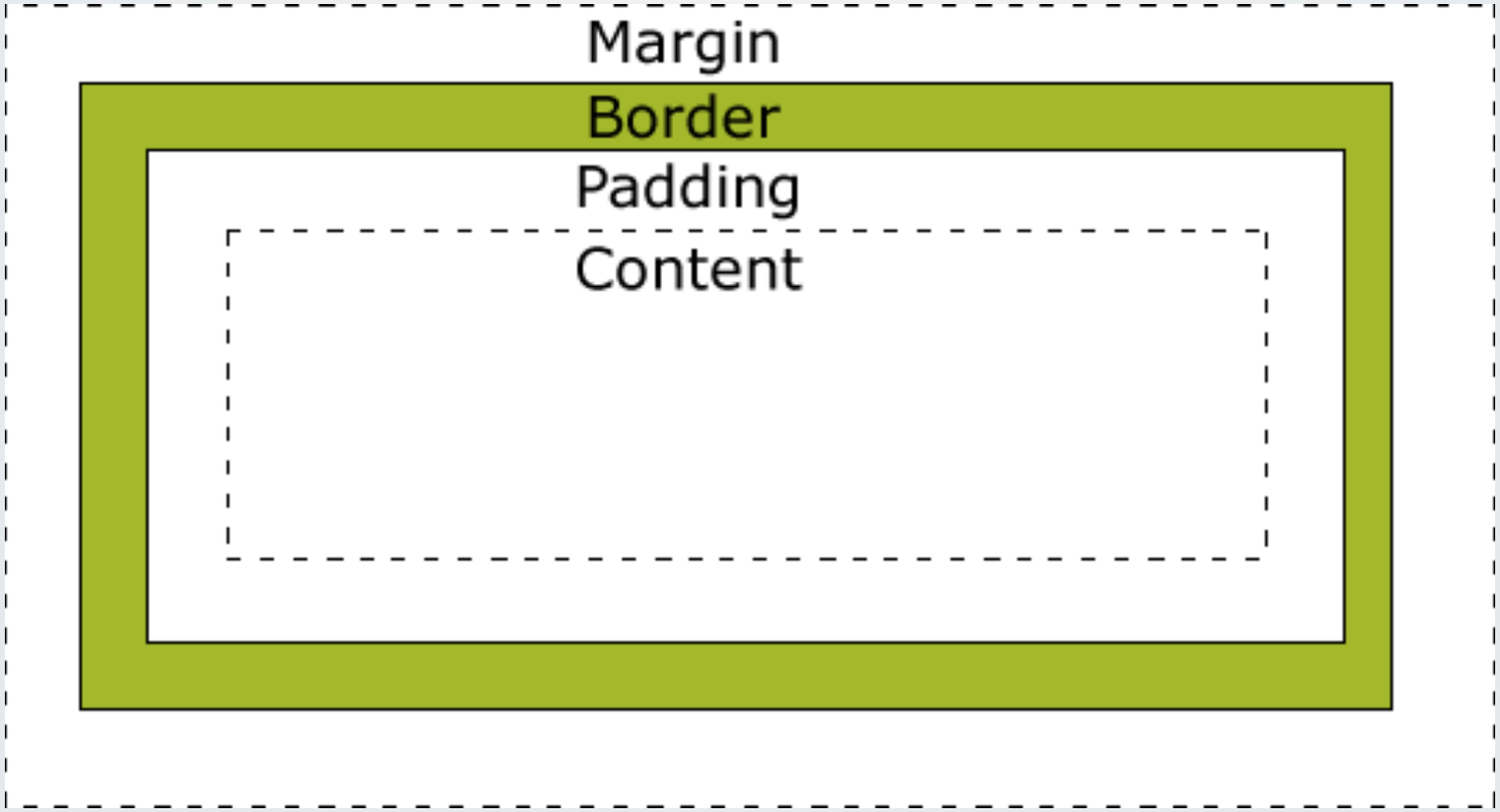
# Setting Upper and Lower Bounds

The max-width and max-height properties can be used to specify the upper bounds for the dimensions of an HTML element.

The min-width and min-height properties can be used to specify the lower bounds for the dimensions of an HTML element.

# CSS Box Model

When spacing HTML elements in CSS we use a box model to specify how to space each element relative to the next.



# CSS Box Model

When you specify the width/height for an element, you are specifying the dimensions of the content (innermost part of box model).

Padding and margin widths are applied afterward, so they will make your element's dimensions larger.

# Padding

Padding is the layer just around the content, before the border.  You can set a uniform padding or set each side independently.

```
div {
   padding-top: 25px;
   padding-right: 50px;
   padding-left: 50px;
   padding-bottom: 30px;
}
```

# Padding

Padding is the layer just around the content, before the border.  You can set a uniform padding or set each side independently.

```
div {
   padding: 25px;
}
```

# Padding

You can also use shorthand and still specify different values for different dimensions.

```
div {
    padding: 25px 50px;
    /* top and bottom are 25px
       left and right are 50px */
}
```

# Padding

You can also use shorthand and still specify different values for different dimensions.

```
div {
    padding: 25px 50px 30px;
    /* top is 25px
       left and right are 50px
       bottom is 30px */
}
```

# Border

Outside the padding is the element's border.
You can specify the style, size, and color.

```
div {
    border-style: solid;
    border-width: 3px;
    border-color: red;
}
```

Available styles: solid, dotted, dashed, double, groove, ridge, inset, outset.

# Border

You can specify different borders for different sides:

```
div {
    border-left-style: solid;
    border-left-width: 3px;
    border-left-color: red;
}
```

You can apply different border rules for top, left, right, and bottom.

# Border

The same shorthand as we used in padding applies for border rules:

```
div {
    border-style: dotted solid;
    /* top and bottom are dotted
       left and right are solid */
}
```

# Border

The same shorthand as we used in padding applies for border rules:

```
div {
    border-style: dotted solid dashed;
    /* top is dotted
       left and right are solid
       bottom is dashed */
}
```

# Border

Special shorthand can be applied if you have a uniform border for all sides:

```
div {
    border: 3px solid red;
}
```

# Margin

The margin is the outermost layer, after the border.  You can set a uniform margin or set each side independently.

```
div {
   margin-top: 25px;
   margin-right: 50px;
   margin-left: 50px;
   margin-bottom: 30px;
}
```

# Margin

The margin is the outermost layer, after the border.  You can set a uniform margin or set each side independently.

```
div {
   margin: 25px;
}
```

# Margin

You can also use the same shorthand as we use with padding.

```
div {
    margin: 25px 50px;
    /* top and bottom are 25px
       left and right are 50px */
}
```

# Margin

You can also use the same shorthand as we use with padding.

```
div {
    margin: 25px 50px 30px;
    /* top is 25px
       left and right are 50px
       bottom is 30px */
}
```

# Positioning

Positioning rules of CSS are some of the trickiest rules to understand (especially when you combine them).

There are four ways to position elements:

- Static
- Relative
- Absolute
- Fixed (the devil)

# Caution

A word of caution…

God save you if you try positioning anything in older versions of Internet Explorer.

Your beautiful layout will fall to pieces.

# Static Positioning

All elements default to the static position. This basically means "put this element where you would normally put it."

```
div {
  position: static;
}
```

It's only useful to specify static positioning if you need to override a rule that says to position differently.

# Relative Positioning

Relative positioning allows you to specify a location for the element *relative* to where it would normally appear.

```
div {
    position: relative;
    top: 10px; /* move down by 10px */
    right: 20px; /* move left by 20px */
}
```

You need to invert your sense of direction a bit.

# Absolute Positioning

Absolute positioning removes the element from the document and lets you specify exactly where on the page to place it (and it goes on top of anything naturally placed there).

```
div {
   position: absolute;
   top: 10px; /* 10px from top edge */
   right: 20px; /* 20px from right edge */
}
```

# Fixed Positioning

Fixed positioning allows you to specify a location for the element to be fixed with respect to the *window* rather than the page.

This means it stays in that position even when you scroll down.

```
div {
   position: fixed;
   top: 10px; /* 10px from top edge */
   right: 20px; /* 20px from right edge */
}
```

# Fixed Positioning is EVIL

Don’t torment your users!

# Floats

CSS “float theory” is the hardest part of CSS positioning.

Floats let you say “I want to shove this div all the way to the left/right,” and have everything else move around it.

```
div {
   float: left; /* shove to the left */
}
```

# CSS SELECTORS AND SPECIFICITY

HACK YALE

# Selectors

Review of selectors we've seen:

| | |
|---|---|
| div { … } | Selects all divs |
| #foo { … } | Selects element with id "foo" |
| .foo { … } | Selects elements with class "foo" |
| div#foo { … } | Selects the div with id "foo" |

*Note: this is unnecessary since id is unique*

| | |
|---|---|
| div.foo { … } | Selects divs with class "foo" |
| div.foo, p { … } | Selects divs with class "foo" OR any paragraph. |

# Combinators

We can use combinators to define relationships between selectors and chain selectors together.

Four different combinators:

- Descendant Combinator
- Child Combinator
- Sibling Combinator
- Adjacent Sibling Combinator

# Descendant Combinator

A space between the selectors indicates that we should match the second selector against elements that descend from those matched by the first one.

Example: any `span` tag that falls within a `div` with the class “green-spans” should be colored green.

```
div.green-spans span {
  color: green;
}
```

# Child Combinator

The child combinator is a special case of the descendant combinator.  It requires that the second selector matches an immediate child of the element matched by the first selector.

Example: any `li` tag that falls within a `ul` with the class “bold-list” should be bold.

```
ul.bold-list > li {
  font-weight: bold;
}
```

# Sibling Combinator

A ~ between the selectors indicates that we should match the second selector against elements that are siblings of those matched by the first one.

Example: any `div` that is siblings with a `p` should have a border.

```
p ~ div {
  border: 1px solid black;
}
```

# Adjacent Sibling Combinator

The adjacent sibling combinator is a special case of the sibling combinator. It requires that the second selector matches the element immediately following the one matched by the first selector.

Example: any `span` that immediately follows an `img` should be italicized.

```
img + span {
  font-style: italic;
}
```

# Specificity Principles

1. The more specific the selector, the greater the specificity.
2. When two rules have equal specificity, the latter wins out.
3. We prioritize rules defined in the HTML document with the `<style>` tag over rules defined in external .css files.

   *You can think of this as part of the second rule: the rules defined in the HTML document are the "last rules."*

# Specificity Precedence

1. Inline styles defined with `style` attribute
2. ID selectors
3. Class / attribute selectors
4. Element selectors

# Specificity Precedence

In cases where you have complex rules that use combinators, we use a formula to determine specificity:

1000(inline?) + 100(#ids) +
10(#classes) + 1(#elements)

When in doubt, there are specificity calculators available online.

# Announcement

Next week we will be moving to a different location.  Likely upstairs or HLH17.

New location will be emailed out when confirmed.

The website will also be updated with the new location.

# Homework

Homework has been posted on the course website: https://github.com/hackyale/Web-Development-101

The fourth assignment can be found under assignments/week_3.md

# HACK YALE

## < INTRO TO WEB DEVELOPMENT />

WWW.HACKYALE.COM