LC 101

Unit 3 - HTML, CSS, JS

March 27, 2017

CSS Precedence

- Styles have the following precedence (from highest to lowest). Higher precedence styles will override lower ones
  - `style` attribute of the tag
  - style applied by `id`
  - style applied by `class`
  - style applied by tag type
  - style inherited from parent elements
    - Only some properties are inherited by default (e.g., `color` is inherited while `border` is not)
- For styles at the same precedence level, those declared later will override earlier ones
  - For class-level styles in particular, it is not the order of the names in the `class` attribute that matters but rather the order of the styles in the page

Block vs Inline

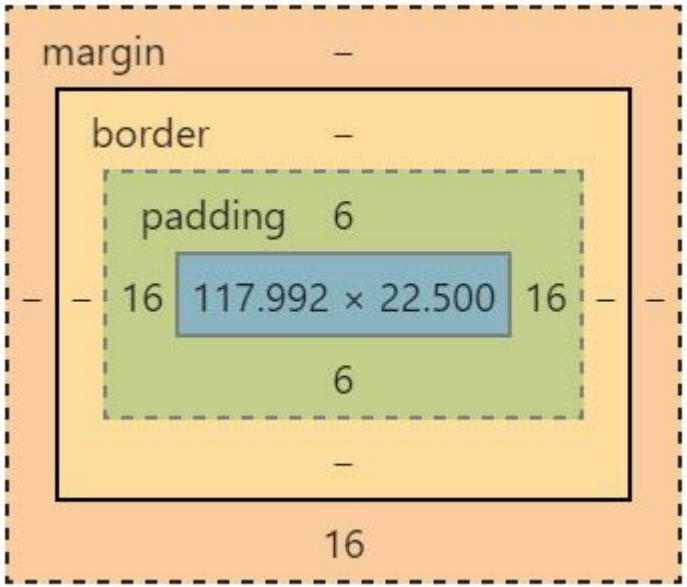
- *Block* elements can be thought of as forming blocks on the page
  - By default, block elements begin on new lines
  - They can contain other block elements as well as inline elements
  - The **`width`** and **`height`** CSS properties affect the size of the block
  - Examples: **`p`**, **`h1`**, ..., **`h6`**, **`ol`**, **`ul`**, **`div`**, **`fieldset`**, **`form`**, **`table`**
- *Inline* elements
  - By default, inline elements do **not** start on new lines
  - They can contain data and other inline elements
    - But usually not block elements
  - Examples: **`a`**, **`br`**, **`img`**, **`input`**, **`span`**
    - **`button`** is unusual; it is considered inline but is generally displayed as *inline-block*
- In HTML5, these categories are being replaced with *content categories*
  - https://developer.mozilla.org/en-US/docs/Web/HTML/Block-level_elements#Block-level_vs._inline

CSS Display Property

- The display property can change how an element is displayed
  - **`display: none`** will hide an element, as if it wasn't there
    - **`visibility: hidden`** will hide an element but still show the space it would be in
  - **`display: inline`** will cause a block element to display as if it was inline]
  - **`display: block`** will cause an inline element to display as a block
    - This only changes how it is displayed, not what kind of element it is. So it still cannot have other block elements inside it
  - **`display: inline-block`** will cause an element to be treated as a block but not start a new line
    - http://learnlayout.com/inline-block.html

The CSS Box Model

- An element is represented as four nested boxes
  - The content is in the center
  - The **`padding`** surrounds the content
  - The **`border`** surrounds the padding
  - The **`margin`** surrounds the border
- The **`width`** and **`height`** properties of an element only refer to the content
  - IE8 and earlier included the padding and border in the width and height



- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model

The CSS Box Model

- **`width`**, **`height`**, **`padding`**, **`border`**, and **`margin`** can all be set via CSS

```
.someClass {
    width: 300px;
    height: 200px;
    border: 5px solid gray;
    padding: 10px;
    margin: 0;
}
/* full width is 300 + 5 * 2 + 10 * 2 + 0 = 330
   full height is 200 + 5 * 2 + 10 * 2 + 0 = 230 */
```

Padding

- Padding for the four sides can be specified individually or in a shorthand property
  - https://www.w3schools.com/css/css_padding.asp

```
.example1 {
    padding-top: 50px;
    padding-right: 30px;
    padding-bottom: 20px;
    padding-left: 0;
}
```

```
.example2 {
    /* same result as .example1 */
    padding: 50px 30px 20px 0;
}

.example3 {
    /* same padding applied to all four sides */
    padding: 10px;
}
```

Padding

- Values can be specified as
  - Fixed length in px, pt, em, cm, etc.
    - Note that 0 does not need units
    - Example: `padding: 10px;`
  - Percent of the width of the containing element
    - Example: `padding: 5%;`
  - Inherited from parent
    - Example: `padding: inherit;`

Margin

- Like padding, margin can be specified individually or in a shorthand property
  - https://www.w3schools.com/css/css_margin.asp
- In addition to fixed size, percent, and **`inherit`**, margin can be specified as **`auto`**
  - **`margin: auto;`** will center the element horizontally in its container
- Margin (unlike padding) can be negative
- The top and bottom margins of adjacent elements are sometimes *collapsed* into a single margin
  - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Mastering_margin_collapsing

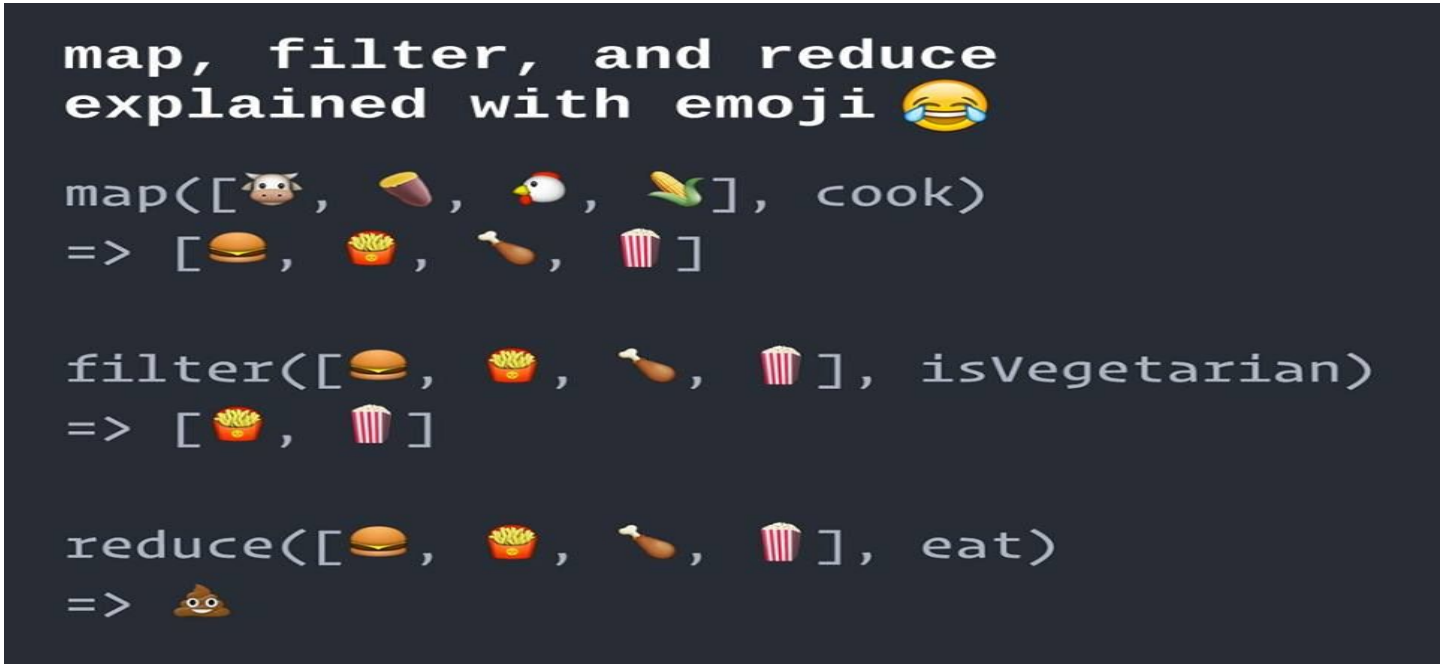
Borders

- Borders not only have width but also
  - style – solid, dashed, dotted, etc.
  - color
  - radius – for rounded corners
- The values can all be specified individually or with shorthand properties
  - https://www.w3schools.com/css/css_border.asp

Semantic HTML

- In HTML4, page structure often consisted mostly of generic **`div`** (for blocks) and **`span`** (for inline) elements
  - Attaching classes to the divs and spans allows for visual styling but provides no semantic information
    - Not ideal for automated processing (search engines, etc.)
- HTML5 introduced new semantic elements for structuring pages
  - **`header`**, **`footer`**, **`section`**, **`article`**, etc.
  - Also introduced multimedia tags like **`video`**, **`audio`**, and **`canvas`**
- Most modern browsers support HTML5
  - IE only introduced full support in version 11
  - https://www.w3schools.com/html/html5_browsers.asp

Map, Filter, and Reduce



- https://i.redd.it/yf7rw3pjiapx.jpg

Array.forEach

- The **`forEach`** method executes a function for each element of an array
  - The supplied function can take three arguments
    - element – the current element being processed
    - index – the index of the current element
    - array – the array itself

```
var someNumbers = [ 1, 2, 3];
someNumbers.forEach(function(element, index, array) {
  console.log('a[' + index + '] = ' + element);
});
```

Array.map

- The **`map`** method creates a new array by applying some function to each element of the original array
  - Like **`forEach`**, the supplied function can take three arguments: element, index, and array

```
var someNumbers = [ 1, 2, 3];
var doubled = someNumbers.map(function(n) {
  return n * 2;
});
console.log(doubled);  // outputs [2, 4, 6]
```

Array.filter

- The **`filter`** method will return a new array containing only those values that pass a test
  - The supplied function should return a boolean value
  - Like **`forEach`** and **`map`**, the supplied function can take three arguments: element, index, and array

```
var someNumbers = [ 1, 2, 3];
var evenNumbers = someNumbers.filter(function(n) {
  return n % 2 == 0;
});
console.log(evenNumbers);  // outputs [2]
```

Array.reduce

- The **`reduce`** method combines the values of an array by applying a two-argument function to an accumulator value and each element of the array
  - The arguments to the reduce method are the function to apply and the starting value for the accumulator
    - The accumulator is optional. If not provided then the first element of the array is used
  - The supplied function can take four argument: accumulator, element, index, and array

```
var nums = [ 1, 2, 3];
var sum = nums.reduce(function(s, n) {
  return s + n;
}, 0);
console.log(sum);  // outputs 6
```

```
var nums = [ 1, 2, 3];
var sum = nums.reduce(function(s, n) {
  return s + n;
});
console.log(sum);  // outputs 6
```