**Lesson 2**

*Developer Tools*

* The inner-structure of the site
* Tools – Developer tools
* Panel in the browser – structure of the page – what you see is not the code but is the structure of the page
* All elements are rectangular
* Not all elements are visible
* You can read the same text as on the page

*HTML’s “tree-like” Structure*

* “Branching” Structure
* Starts with a root Element and branches from the root to child elements
* All elements can have sub elements (child elements)
* How does the browser know what the structure is and how to display it?
* How HTML Classifies page content
* Tells browser what is an image, text, and so on
* DOM

– Document Object Model - a cross-platform and language-independent convention for representing and interacting with objects in HTML (and other markup languages). The nodes of every document are organized in a tree structure, called the DOM tree.

*Indentations and Boxes*

* CSS manipulates shapes seen on webpage
* The element contained
* Unchecking border-radius changes circles into square
* Turning circles into Squares – power of style – everything on a webpage is a *box*
* Easy to re-arrange things because of boxes
* Grid in the background make sure everything is properly aligned
* Give an element a class attribute – where the element will go – can belong to multiple classes
* Names - title app screenshot description

*Text Editors*

* When writing code, programmers use special text editors (like Sublime Text for example). These editors make the programmer's life easier. For example, some text editors will automatically generate a closing HTML tag when you write an opening tag.

**Lesson 3**

*Avoiding Repetition*

-

*CSS*

* Cascading Style Sheets
* You can also think about CSS as a search and replace tool: you identify a class or a tag of the element you want to find (or match, in CSS terminology), and then what you want to do with it, or what property values to replace with different ones.
* Style sheet language used for describing the look and formatting of a document written in a markup language
* **Stylesheet in a separate file (mostly used)**
* **Stylesheet inside HTML (small projects but is not ideal)**
* **Inline style in an element (should be avoided)**
* Css selector – defines what elements what a particular style will apply to

.selector {

property: value; (declaration)

}

* CSS is code written to control “style” of HTML elements.
* **Rule**: a line of CSS code describing the value that a certain attribute should take.
* **Property**: The property you want to change.
* **Value**: The value that you want to assign to the attribute.
* **Selector**: The name that you use to in order to target the elements that are assigned to a class or id attribute in the HTML.
* **Class**: A class refers to a group of elements that can be styled together. Class names should not contain periods or any other punctuation marks such as class="1.1"
* **ID**: ID's are unique identifiers that uniquely identifies an element in HTML.
* Example using word documents - “Tell what part of the document the user is reading”

*Coding Resources*

W3schools - http://www.w3schools.com/default.asp

Normalize.css - <http://necolas.github.io/normalize.css/>

Mozilla Developer Network - <https://developer.mozilla.org/en-US/docs/Web/CSS>

Extra information…

The Box Model

Before positioning boxes, get a better understanding of *The Box Model…*

**Margin** – clears area around the border – space between boxes – no background color – transparent – (**change margin to push text away)**

**Border** – around padding and content – inherited by color property of the box

**Padding** – Clears an area around content, effected by background color of the box – space inside the box that protects content

**Content** – “nucleus” – padding, border, margin protect content so other items don’t overlap or sit to close

* Size of actual element is equal to **border**, **padding** and **content** width.
* This makes it hard to know how much space the box will take on the screen without recalculating it each and every time you change one of these values
* Easier used model has been developed - Box-sizing: border-box;

CSS style

Box-sizing: border-box;

Browser specific prefixes:

-webkit-box-sizing: border-box;

-moz-boxsizing: border-box;

-ms-box-sizing: border-box;

NOW add max width attributes to the divs encompassing the images and description of the app

* Makes calculation of the elements size include both the border and padding, **so it’s easier to make layouts**
* Set the width of overall box
* No matter how you tweak padding & border sizes (width and lengths), the size of the box will always stay the same
* *Margin* is not included in the size
* *Drawback –* developed recently, make sureolder browsers are able to display it, need to add: browser specific prefixes
* Add in front of style definition
* Setting size in pixels and percentage
* Percentage – size will change depending on how big the screen is or how encompassing the box is – BEST when resizing box
* Pixels – take exact size – not change no matter the size of the screen – can lead to a bad user experience
* Box size changes/layout stays the same

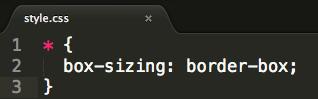
**Positioning Boxes**

**FLEXBOX LAYOUT –** Flexible box

* Display: flex;
* Provides efficient way to lay out a line
* Distribute space among items in a container (div)
* For this to work, need to size smaller than the 100%
* This rule as-is works with Chrome 29+, IE 11+, and Mozilla 28+
* To work in Safari add – *display: -webkit-flex;*

IN SUM

Four main points about box sizing

1. HTML elements are boxes and each box has 4 components.
2. Because there are so many components to each box, it can often be hard to get the size of a box just right.
3. There are two techniques you can use to help deal with sizing issues.
4. Set sizes in terms of percentages rather than pixels.
5. Set the box-sizing attribute to border-box for every element. 
6. Different browsers work slightly differently. Sometimes this causes different browsers to display the same code differently.

### Box Positioning

1. Divs are **block** elements (as opposed to **inline**), so by default they take up the entire width of a page.
2. Adding the rule display: flex; to the appropriate CSS will override this behavior and let divs appear next to each other.

Display: flex;

<div class=”image”><img src=”images/app.png” alt=”This is a screenshot”></div> - **display flex image next to text!**

Code, Test, Refine

Testing and modifying

How do you know where to start testing?

What do you refine?

Get structure of the page right

Tags

Boxes size

Postioning

Then…

Smaller details

**Solution:**

1. Look for natural boxes
2. Look for repeated styles & semantic elements
3. Write your HTML
4. Apply styles (from Biggest to smallest)
5. Fix the small things

**DevTools**

* Helps code, test, refine
* Able to test things on the fly
* How it affects your page without changing html & css files
* Change font-sizes, font-family, background colors using slider, etc.
* Sources tab – see the code in the same place
* Reset changes using refresh button
* Refine it on dev tools then save it to css file \*\*important\*\*

### Verifying HTML and CSS

To verify HTML: <http://validator.w3.org/#validate_by_input>  
To verify CSS: <http://jigsaw.w3.org/css-validator/#validate_by_input>

HTML and CSS is correct and complies with standard

* Correct in one browser
* Incorrect in another – different assumption

**Lesson 2: Creating a Structured Document with HTML**

**Developer Tools (in the Browser)**

HTML elements are either **inline** or **block**. Block elements form an "invisible box" around the content inside of them.

**The "tree-like structure" of HTML**

The "tree-like structure" comes from the fact that HTML elements can have other elements inside of them. You can draw this relationship like a family tree. My mother had multiple children. So did her mother, and so on...   
In a browser, this structure shows up as a series of nested boxes. There are boxes inside of boxes inside of boxes, and so on...

**The relationship between indented HTML and boxes**

When you read an HTML document as *text*, you see a wave of changing indentations going up and down the left side of the document. The more indented an element is, the more deeply nested it's corresponding "box" is.

**Text Editors (for programming)**

When writing code, programmers use special text editors (like Sublime Text for example). These editors make the programmer's life easier. For example, some text editors will automatically generate a closing HTML tag when you write an opening tag.

Span can edit texts within a paragraph – can be put inline

Divs – does a line break