# Section 1 – First steps

## High-level overview of Web Development

Static website 🡪 Upon request to the server, the website assets are sent to browser as it is

Dynamic website 🡪 Upon request to the server, the backend will assemble the website’s assets and then send to the browser

# Section 2 – HTML Fundamentals

## Document Structure

<!DOCTYPE html> <=== Declare that this document is using HTML

<html> <=== A HTML element to contain all the stuff the HTML doc should have

<head></head>

<body></body>

</html>

## Text elements

|  |  |
| --- | --- |
| <h1> to <h6> | Helps to break up text content into sections  Has head from size h1 to h6 (Descending size)  Good practice to have only 1x h1 heading |
| <p> | Paragraph of text |
| <b> OR <strong> | b: bolding text content (OLD WAY OF BOLDING)  strong: same as b (NEW WAY, makes more semantic sense) |
| <i> OR <em> | i: italicise text content (OLD WAY)  em: same as i (NEW WAY, makes more sematic sense) |
| <ol> | Ordered list |
| <ul> | Unordered list |
| <li> | List item |

## Semantic tags

|  |  |
| --- | --- |
| <header> | Contains introductory content for a page (e.g., a banner), or a section of a page. |
| <nav> | Contains navigation content, such as a website navigation menu. |
| <main> | Contains the main content of the web page. |
| <aside> | Contains content that is tangentially related to the main content of the page (often this is presented in a sidebar). |
| <footer> | Contains the footer of a page, or of a section of a page. Typically, the footer contains information about the content, such as the author and a copyright statement. |
| <article> | Represents a self-contained composition in a document, page, application, or site, which is intended to be independently distributable or reusable |

## Semantic HTML

A way of writing HTML whereby each HTML element is writing with the appropriate tag with a proper meaning. This allows clear communication of each element’s meaning to the developer, browser, and users.

* Improve accessibility as text to speech programs make use of the HTML tags to recognize contents

# Section 3 – CSS Fundamentals

## CSS Rule

Many of these can be within 1 .css file

Diagram, timeline

Description automatically generated

## Inline vs Internal vs External CSS

|  |  |  |
| --- | --- | --- |
| **Inline** | **Internal** | **External** |
| CSS that are within the opening tag of a HTML element | CSS that are within the <head> of a .html file | CSS that are in its own isolated .css file and included into the .html file |
| Eg.  <h1 **style=”font-transform:uppercase”** /> | Eg.  <head>  **<style>**  **h1 {**  **font-transform: uppercase;**  **}**  **</style>**  </head> | Eg. **style.css**  **h1 {**  **font-transform: uppercase;**  **}**  index.html  <head>  <link href=”style.css” rel=”stylesheet”/>  </head> |

## Selectors

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Description** |
| Element Selector | h1 {…} | Select elements by its type |
| Class selector | .primary-header {…} | Select elements by its class |
| ID selector | #author {…} | Select elements by its ID |
| List selector | h1, h2, h3 {…} | Select elements that satisfying any selector included above |
| Descendent selector | article p {…} | Select the child element of an enveloping element |

## Pseudo-classes

Keywords that can be added to a selector to specify the state of the selected element you are looking for

* Eg. *li:first-child*: Select a list item element that is the first child element of its parent element

## Hyperlinks

When styling an anchor (ie. hyperlink) element, we must take care of its 4 states:

* :link, :visited, :hover, :active
  + :link -> Selects anchor that are actually a link (ie. has href attribute)
  + :active -> Selects anchor that are currently being clicked
* The element is expected to be styled differently for each state and the styling should be done in that order

## Theory 1 – Conflicts between Selectors and Stylings

When there are >1 selectors selecting an element, all of them applies to it

For any rule that is in conflict (ie. font-size: 10px; and font-size: 15px;), resolve them with the following diagram:

Graphical user interface, text, application, chat or text message

Description automatically generated

* For selectors of differing priorities, the highest priority wins.
  + The more specific a selector, the more likely the styling gets the higher priority
* For selectors of the same priority, the latest selector wins.

## Learning points in Challenge #1

* Use **hexadecimal to represent colors**
* **text-transform**: uppercase
* **text-align** not align-text
* **text-decoration**: underline
* **Use classes whenever possible**, don’t be too general
* **list-style-type** to adjust the bullet point shapes (PUT IN LIST INSTEAD OF LIST ITEMS)
* **cursor: pointer** to change cursor to pointing finger when hovering a button

## Theory 2 – Inheritance and Universal Selector

### Inheritance

Certain properties from the parent elements are passed down to the child elements

* Mostly properties related to text
  + font-family, font-size, font-weight, font-style, color, line-height, letter-spacing, text-align, text-transform, text-shadow, list-style, etc.
* Inherited properties have the lowest priority, easily overwritten

Graphical user interface

Description automatically generated

This means that we **can propagate styling downwards in a similar fashion as event delegation** in JavaScript

## Universal Selector

\* {…}

Allow us to apply style to every element in the page but without any inheritance happening

* Useful for applying properties where it cannot be inherited
* Opposite of applying to *body* to let the properties be inherited by child elements

## Theory 3 – CSS Box Model

Defines how elements are displayed on a page and how they are sized. Each element on the page can be seen as **a rectangular box that has content, a border and space inside and outside of it**:

Diagram

Description automatically generated

* **Padding**: Spacing between the content and the border (Internal spacing)
* **Margin**: Spacing between the border and the nearby elements (External spacing)

### Box model – content-box

Diagram

Description automatically generated

* *Box-sizing: content-box*
* All the above values can be modified via CSS

### Box model – border-box

Defined width and height include the padding and borders of the box (ie. Whatever width and height that are defined, it will be the final width and height)

Diagram

Description automatically generated

* *Box-sizing: border-box;*
* Most CSS developers just set this to all elements using the \* universal selector

### Setting dimensions (width & height)

Width & height can be set to several units:

|  |  |
| --- | --- |
| Px | Pixels, doesn’t scale with window size |
| % | Percentage of the parent element’s height |
| auto | Scale automatically with the element’s content and other properties |

**Note**. Remember that these heights are just the defined dimensions not the final dimensions.

### Margins and Paddings

#### Global reset

A common practice to reset all paddings and margins to 0 using the universal selector

\* {padding: 0; margin: 0}

🡪 Provide a clean slate for the developers to work from and give explicit paddings and margins

#### Collapsing margins

When neighboring elements have margins that overlapped with one another, only the larger margins will be displayed and the smaller one will be ignored

* Eg. margin: 40px vs margin: 15px => margin: 40px will be displayed

## Trick to center a page’s content

1. Encapsulate the content into a blank container element like a <div>
2. Style the div to have
   1. Fixed width (Stop content from stretching)
   2. Auto left-right margin (Automatically balanced the left and right margin, giving the illusion of centering the container)

## Theory #4 – Types of boxes an element can be in

An element can be displayed in 3 different types of blocks:

Graphical user interface, application, Word

Description automatically generated

* By default, an element uses either block-level or inline boxes
* Can be modified but in normal use cases, shouldn’t modify

## Theory #5 – Absolute positioning

An element has 3 forms of positioning: Normal, Relative, and Absolute

Graphical user interface, application

Description automatically generated Diagram

Description automatically generated

* **Normal position**: Default positioning, laid out according to the order in the HTML code
* **Relative position:** 
  + Like normal position but it let the element to serve as an anchor for absolute position elements
* **Absolute position:**
  + Remove the element from the standard flow of positioning
  + Can no longer have an impact on surrounding elements
  + Uses top, bottom, left or right to offset from the closest relatively positioned container element (Like an anchor)

## Pseudo-element

Elements that don’t exists in the HTML but that we can still select and style in CSS

**Examples**

::first-child, ::first-letter, ::before (Creates an element as the first child for the selected element), ::After (Similar to ::before but as the last child for the selected element)

# Extracts from the Advanced CSS Course

## What happens to HTML and CSS upon page load

Diagram

Description automatically generated

When the HTML and CSS are loaded, they are parsed to form their respective object models

* When parsing a CSS, there are mainly 2 processes
  + Resolving conflicting CSS declarations (Cascading)
  + Processing final CSS values

## Cascading

A process of combining multiple stylesheets together and resolving the conflicts it has between different CSS rules and declarations when they are applied to an element

* Ie. Declarations from a CSS rule conflicts with declarations from another CSS rule, we need cascade to decide which declaration the element will use

### Sources of CSS declarations

|  |  |
| --- | --- |
| **Author** | Declarations written by the developers within the .css files |
| **User** | Declarations written by the users outside of the .css files  (eg. Declaring in Chrome devtools) |
| **Browser** | Default declarations that a browser made as a baseline |

### Cascading flow

A screenshot of a computer

Description automatically generated with low confidence

#### Importance

* Author declarations & User declarations are basic declarations without any !important

#### Specificity

Specificity of an element is determine based on how many different types of selectors were used in the CSS rule:

**(# of inline styles, # of IDs, # of classes/pseudo-classes/attribute, # of elements/pseudo-elements)**

We then compare the # of selectors of each type between the conflicting declarations. The declaration with higher number of more specified selector wins the conflict.

Eg.

Text

Description automatically generated

# Section 4 – Layouts: Floats, Flexbox and Grid

## 3 ways of building layouts

1. Floats
2. Flexbox
3. Grid

## Floats

A CSS property to remove an element from the normal flow, but allow other elements to wrap around it (Unlike absolute positioning)

Note. If all elements in a container float, the container element will **collapse**

### Clear float

A CSS property that set whether an element must be moved below floating elements that precede it

* Can be applied to floating or non-floating elements
* Element that is being cleared must be a block element
* Can be applied to a pseudo-element if there isn’t any element to keep the container element stretched

## Flexbox

A set of related CSS properties for building 1-dimensional layouts

* Based on the idea of distributing empty space in a container element to its child elements
* Make floats obsolete

### Components of a flexbox

Diagram

Description automatically generated

* *flex-direction* determines which axis will be the main axis and cross axis
  + Affects justify and alignment of items

### Properties

Graphical user interface, text, application

Description automatically generated

#### Flex shorthand

Text

Description automatically generated

* *flex: 1 ⬄ flex: 1 1 0 ⬄* All flex-items has the same size regardless of content

## Grid

A set of CSS properties for building 2-dimensional layouts

* Based on the idea of dividing a container element into rows and columns that can be filled with child elements
* Not meant to replace flexbox
  + 1D layout: flexbox
  + 2D layout: grid

### Components of Grid

Graphical user interface, application

Description automatically generated with medium confidence

* Unlike flexbox, the grid’s axis cannot be changed
* Gutters ⬄ Row gap or Column gap
* Each grid line is labelled with a positive and negative line number, both count from opposing directions

Graphical user interface

Description automatically generated

### Properties

Graphical user interface, text, application

Description automatically generated

#### Fr & auto

fr 🡪 A proportion value that we can give to distribute free space across a grid’s rows or columns

auto 🡪 Specify that a row/column will take the exact amount of space needed for its content

#### Repeat(…)

For grid-template-columns and grid-template-rows, if we have columns/rows with same distribution, we can declare them one-shot as follows:

#### Spanning a grid item across a row or column