# 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)

### Actual height & width of an element

Diagram

Description automatically generated

* All the above values can be modified via CSS
* The behavior behind computation of width and height can be modified but not recommended

### 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)