Week 4 Notes

STYLING HTML USING CSS

HTML , CSS, & JS

* HTML gives us structure (which is essential)
* CSS gives us presentation
  + Not just making things look pretty
  + But important to usability and accessibility
* Presentation from a technical perspective
* Styling individual elements, arranging groups of elements & bring everything together to create cohesive designs

HTML

* Early HTML mixed content and presentation
  + Every element needed font, colour, style, alignment, border, size, etc. explicitly described, often repeatedly so throughout HTML
  + Moving style declarations gives simpler HTML and more manageable design
* Needed a consistent & flexible way to control presentation: enter CSS

CONTENT & PRESENTATION

* Visual & design aspects separated from the core content & structure of the document
  + Think of human body:
    - Skeleton gives structure
    - Flesh gives appearance
* Not a rigid rule but more of a best practice:
  + Different members of a team can work on each aspect
  + Content can be presented in different ways
    - People with visual impairment can provide their own stylesheets to the browser to override the website developer’s decisions
    - Screen & Print versions of pages can be styled differently - tailor content to the medium of consumption
  + You can use inline presentation elements (or mix inline & separate) but can lead to future problems:
    - Separation leads to simplified change management
  + Design first or content first?
  + Generally you can’t design something well until you have some idea of what the design is working with - you need some content (or at least an idea of the structure of the content)
    - Content can drive the design process
      * One reason why we started with HTML and thinking about Data before getting to CSS
  + Relate to other design ideas:
    - “Form follows function” - how something is structured stems primarily from the underlying engineering requirements
  + We should not conform structure of content for design reasons if it leads to a compromised system, e.g. buggy, inefficient, unreliable, unusable
  + But we can let structure inform the design process – many elegant solutions stem from the design exposing underlying structure

CASCADING STYLE SHEETS (CSS)

* A simple, text-based, page appearance description language
* More strictly: A Declarative, domain-specific, programming language
  + Declarative (implicit control flow) - say what we want & the computer tries to achieve that (& other thinks might happen (e.g. increase font-size & the associated container will grow automatically)
* Resilient - If browser can’t do what you declare in CSS then the site (generally) continues to function
* Permit almost every HTML tag to be arbitrarily scaled, positioned, & decorated – overcomes limitations of underlying markup language
* Don’t think of this as pixel-perfect user interface design
  + not primarily to build UI for web replacements of desktop/mobile apps
  + but as flexible and robust way to share information, primarily as documents, across many platforms
* Moved from (pre CSS) HTML presentational attributes:
  + <h1><font color=“red”>The Quick Brown Fox</font></h1>
* to style parameters:
  + <h1 style="color: red;”>The Quick Brown Fox</h1>
* to style blocks:
  + Text, letter

    Description automatically generated
* to external style sheets:
  + <link href="path/to/file.css" rel="stylesheet" type="text/css">

USING CSS & HTML TOGETHER

* CSS can be used with HTML in 3 primary ways:
  1. Attached to a specific tag using the style parameter
  2. Inlined globally using a <style> block
  3. Retrieved from an external URL using <link  
     rel="stylesheet" type="text/css" href=“theme.css”>  
     directive
* 2 & 3 require a fully qualified stylesheet consisting of any number of selectors

USING THE STYLE PARAMETER

* “inline” with HTML Element attributes, e.g.
  + <p style=“color:red”>
  + Advantage:
  + Put it right where you’re using it
  + Disadvantage:
    - Lots of repetition - have to specify everything
    - Mixture of content & presentation

USING A <STYLE> BLOCK

* Collect all styles declarations together
* Separates presentation from content
* Styles individual pages (but not sites)
* Text, letter

  Description automatically generated

EXTERNAL STYLESHEET

* Separation of content (in .html file) from presentation (in .css file)
* Can reuse the same .css file in multiple .html files
  + So can style an entire site (multiple HTML files) with all the presentation in one place
    - Easy to manage & update the design without touching the content
    - Can add new content & take advantage of ready made style
* NB. media attribute lets you specify different style sheets for different contexts, e.g. print, projection, aural, braille, tty, tv
* Text, letter

  Description automatically generated

BASIC CSS SYNTAX

* Stylesheets are a list of rules
* Rules are constructed from selectors & declarations
* Rules use a simpone or more selectors identify which html elements to target then a declaration that says how to present those elementsle syntax:
  + Timeline

    Description automatically generated

SELECTORS

* Tells the browser which part of the markup to apply a style to, e.g. matching style to HTML tags and attributes
* Can apply to:
  + All elements of a specific type, e.g. <h1> or <img>
  + Elements specified by attribute but often
    - specified by id attribute, e.g. <p id=“para-123”> #para-123
    - specified by class attribute, e.g. .photos selector applied to tags with <p class=“photos”>. .photos
    - NB. pseudo-classes refer to special states of selected elements, e.g. hover, visited, active
      * Enables elements to be styled in relation to things outside the DOM, e.g. history of navigation
  + Elements depending upon their placement relative to others in the DOM
  + Can be combined and joined in many ways to specify elements by location, element type, id, class (or any combination thereof)
  + Order of selectors is important

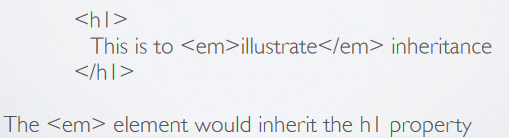
DECLARATIONS

* Works with the selector - tells the browser the set of properties to apply to the elements selected by the selector
* A Declaration comprises a property, a colon (:), and a value, e.g.
  + color: red
* Properties defined in CSS standard & has a given set of values (keywords like “center” or values, e.g.
  + Colour can specified with keywords, e.g. red, or Hex values (#FF0000) or RGB values (rgb(255, 0, 0))
* Get used to consulting documentation:
  + Some CSS properties can affect any type of element. Others apply to particular groups of elements.
  + Each CSS property can take only a specific range of values
* Multiple declarations are separated by semi-colons (;)
  + color: red; text-align: center;
* A Declaration block is a list of declarations in braces
  + h1 {color: red; text-align: center;}

CSS DESIGN CONSTRAINTS

* Applies styling & layout to HTML elements - can paint border around HTML element but cannot create in pure CSS
* Layouts of content that can adjust to any screen/viewport/canvas on any platform
* Text that scales independently of viewport - scaling font should lead to reflow of page not to horizontal scrolling
* User can override anything regardless of what the designer thinks or intends
* Progressive rendering - layout depends on previous content(but not future content) - but loading new content might force a reflow of previously rendered content

INHERITANCE

* A key CSS feature
* HTML parsed into DOM
* DOM is a tree which is hierarchical
* Nested descendants generally inherit text-related properties from parent elements that enclose them
* Efficient because don’t have to declare same properties repeatedly given
  + h1 { color: purple; }
  + But no declaration for the colour of the <em> element
    - 

COLOURS

* An easy place to start, visual impact.
* Control colour of element using the colour property
* 140 named colours
* Each can be referred to by name, hex code, RGB or HSL code
* More colours available by code than are named (~ 16M+)

BACKGROUND

* background-color
* background-image
* background-repeat {repeat, repeat-x, repeat-y, no-repeat}
* background-position - specify two values for horizontal & vertical from {top, bottom, left, right, center}

FONT PROPERTIES

* After colour, typography is an important consideration
  + font-family - which font to use
  + font-size - how big it should be
  + font-style - {normal, italic, oblique}
  + font-weight - {normal, bold, bolder, lighter, +numeric values}

TEXT PROPERTIES

* text-align - justify blocks of text, e.g. {left, right, center, justify}
* text-decoration - {none, underline, overline, line-through, blink}
* text-transform - {non, capitalize, uppercase, lowercase}
* NB. Letter spacing, word spacing, line-height

LINKS

* Four states:
  + link - the normal state
  + visited - if the link has previously been followed
  + hover - while pointer is over link
  + active - whilst being clicked
* Define how links should look in above states, e.g. a:link {color:red}

BOX MODEL

* Diagram, table

  Description automatically generated
* Each element of a web page has several boxes associated with it that affect spacing & placement:
  + Padding - distance between edge of element and it’s content
  + Border - frontier between padding and margin
  + Margin - distance between edge of element and adjacent element
* padding, border, & margin divided into four edges:
  + top, bottom, left, & right:
    - border-left, border-right, border-top and border-bottom
  + Same for margin and padding...
  + Borders can be applied to all edges or to individual edges
  + Border characteristics: {solid, dotted, dashed, double}
  + Border width using a supplied value & colour

LAYOUTS: HTML+CSS

* You can use HTML <span> and <div> block elements to assemble page layouts
  + span - inline, div - block
* Wrap these around the different blocks of content for your page
* Give each a unique id attribute so that they can be identified, referenced, and subsequently positioned & styles by CSS
* NB. HTML requires IDs to be unique
* **Best to use semantic HTML tags** to achieve this (but what if you have groupings that don’t fit to standard semantic tags? (e.g. for columns of content in a multi-column layout for ultra-wide monitors)
* Gives us the basics for assembling layout patterns, e.g.
  + Two column layout: navbar + content
  + Three column layout: navbar + content + sidebar
  + NB. Consider header & footer as well

CSS LAYOUT

* How the browser lays out HTML elements follows from these constraints, i.e. from the display property:
  + inline - Elements flows horizontally left to right as space allows. Line breaks happen when the edge of the container is reached, and the flow continues on the next line.
* block - Elements flow top down, one after another.The width of the block is what is available in the container, the height is defined by the content.
* table - Child elements are aligned both vertically and horizontally.
* flex - Child elements are placed on a single axis either vertically or horizontally, which specified alignment, adjustments and spacing.

PERCEPTION OF CSS

* Perceived as being hard - but often this is due to thinking of CSS as something that it’s not . We want to develop sites that are of the web not merely put on the web
* Properties interact and can lead to unexpected results - set one and it combines with others (& defaults)
  + Never be more explicit than necessary
* HTML web pages are responsive by default so work with this rather than against it or forcing different/overridden behaviour
* Let content determine either width or height (or both) instead of forcing dimensions
* As soon as you try to impose your will on how your content must be viewed then you are setting up problems for yourself