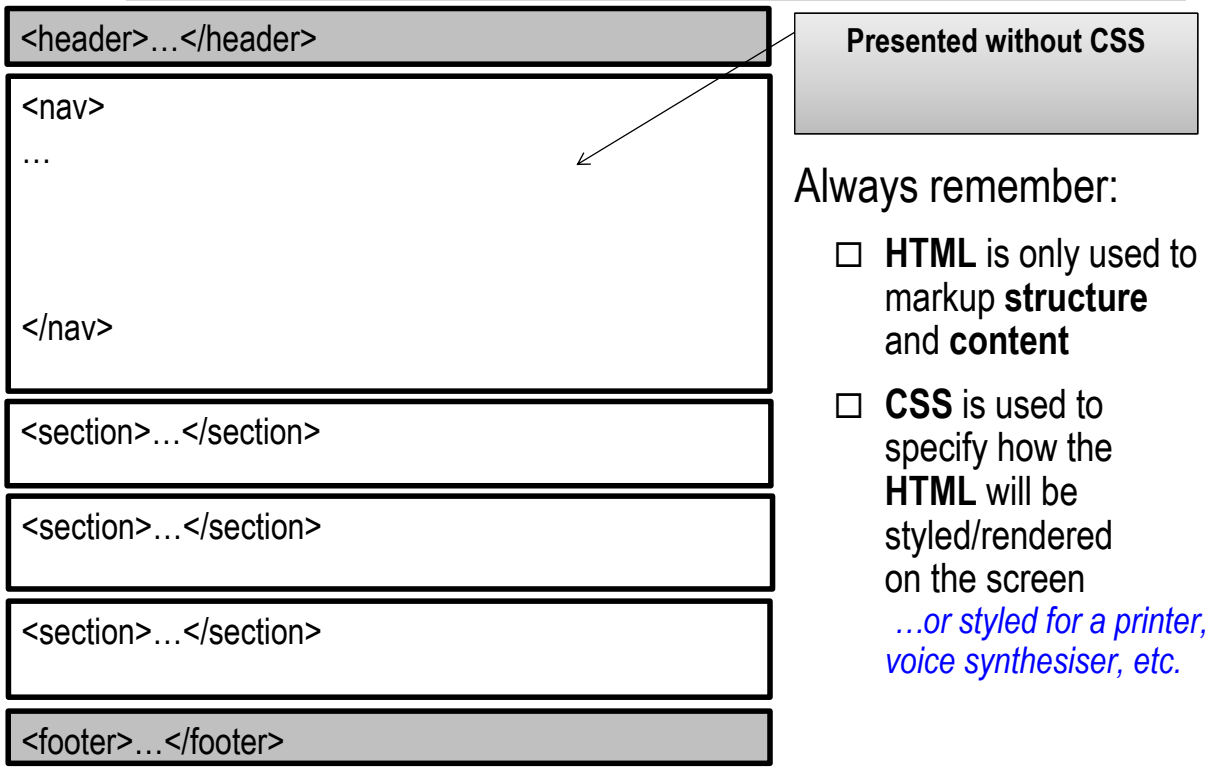
SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10011/60004
Creating Web Applications

Lecture 04
CSS 2
Layout and Responsive Design

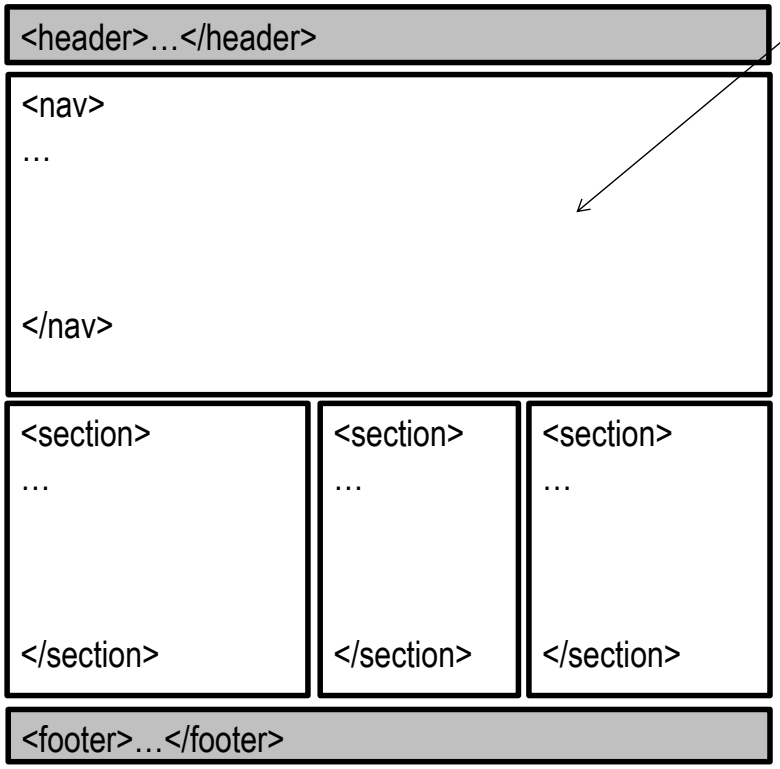
HTML5 structure



Always remember:

- **HTML** is only used to markup **structure** and **content**
- **CSS** is used to specify how the **HTML** will be styled/rendered on the screen
  …or styled for a printer, voice synthesiser, etc.

Applying CSS



How do we make the HTML page know about the CSS?

Add a link in the HTML

Contents

CSS Part 2 Layout

- In-line and Block elements
- Box model
- Page Layout
- Responsive Design
- CSS Frameworks and Preprocessors

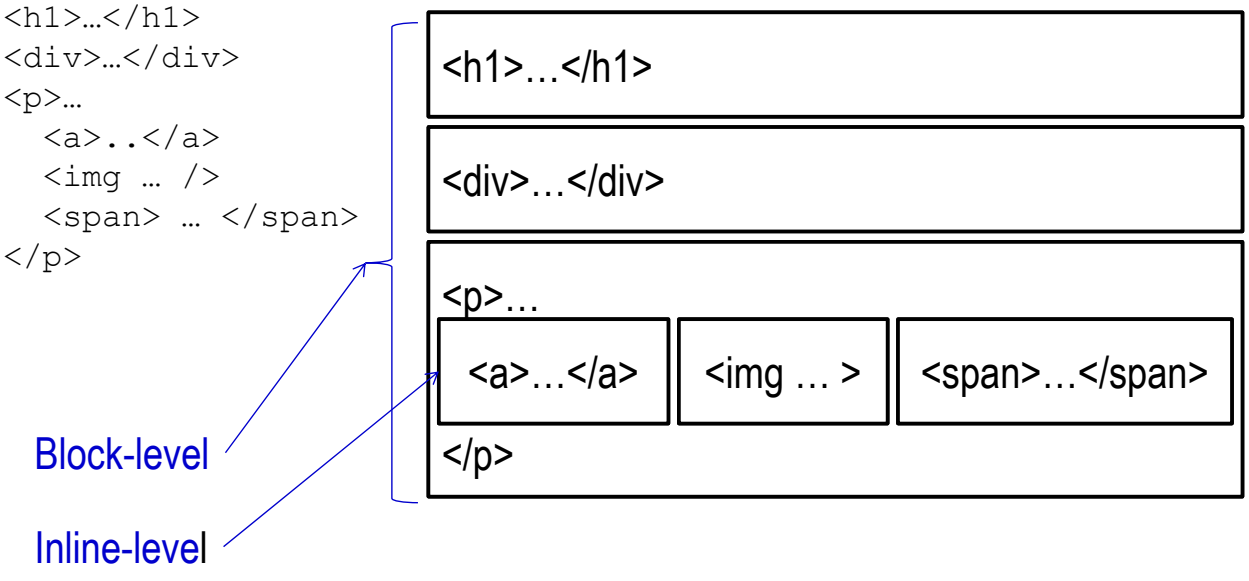
CSS: Visual Format and Box Models

- ***Visual formatting*** model describes how the element content boxes should be displayed
  - **Block-level elements** appear as blocks *such as headings, paragraphs, tables, lists*
  - **Inline-level elements** are contained *within* block-level elements, *such as anchors, images*
- ***Box model*** describes the rectangular boxes that contain content on a web page

Model: Visual Formatting

- Arrangement is top to bottom left to right according to how the HTML elements are ordered

```
<h1>…</h1>
<div>…</div>
<p>…
  <a>..</a>
  <img … />
  <span> … </span>
</p>
```



CSS: Changing inline to block, and more …

- **display** : **inline** | block | list-item | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none
  - **display: block**
    used to change an inline element to a block level element,
  - **display: inline**
    used to change a block level element to an inline element
  - **display: table values**
    used to create table-like displays using CSS
    (HTML tables are only for tabular data)
  - **display: none**
    value hides the element from display

Useful for aligning <li> menu items horizontally

e.g. you might want to hide some elements in a small mobile layout

http://css-tricks.com/almanac/properties/d/display/

CSS: Inline Text Alignment, and other properties

- The text-align CSS property describes how **inline** content like text is aligned ***within*** its parent block element. text-align does not control the alignment of block elements, only their **inline** content.
- **text-align** : **left** | right | center | justify;
  - text-align : center;
    - Justify is not supported by all browser
- **text-indent**: <value>; (indents first line of paragraph) default is **0**
  - text-indent : 2em;
  - text-indent : -2em; (for hanging indent)
- **line-height** : **normal** | <value>;
  - line:-height : 150%; (1.5 spacing assuming font size is normal)
- **text-decoration**: **none**, underline, overline, line-through
- *Also see CSS3 text-decoration:*
  http://www.w3.org/TR/css-text-decor-3/

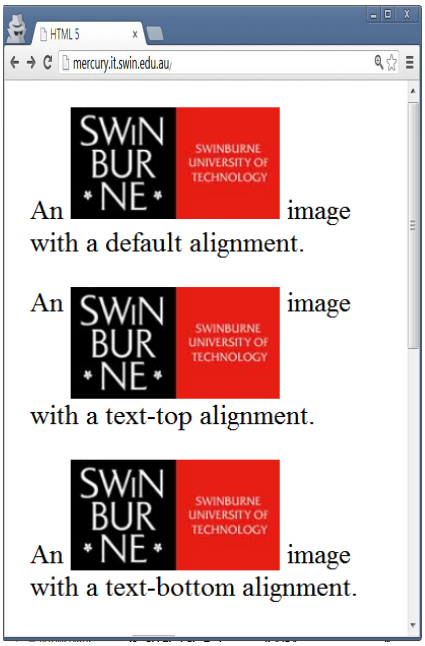
Default *values* shown in **red**

'underline' is default for 'a' element

CSS: Inline Alignment of Graphics with Text

```
<p>An <img src="logo.jpg" alt="SUT" width="120"
height="64" />
image with a default alignment.</p>

<p>An <img class="top" src=" logo.jpg"
alt="SUT" width="120" height="64" />
image with a text-top alignment.</p>

<p>An <img class="bottom" src=" logo.jpg"
alt="SUT" width="120" height="64" />
image with a text-bottom alignment.</p>

</body>
```

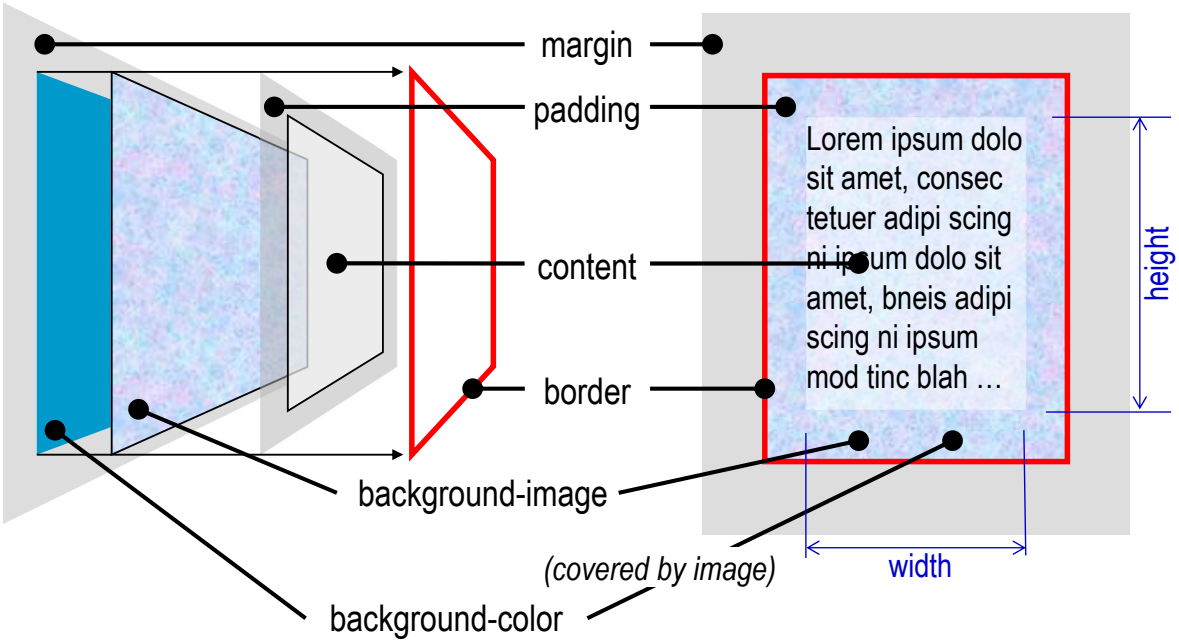
```
CSS
  img.top     {
        vertical-align:text-top;}

  img.bottom  {
        vertical-align:text-bottom;}
```



vertical-align : **baseline** | sub | super | top | text-top | middle | bottom | text-bottom

The CSS Box Model for Block elements

- Below is a representation of the CSS box model.



- *References:*
- CSS Backgrounds and Borders Module Level 3 http://www.w3.org/TR/css3-background/

CSS: Block element Background - Properties

- CSS box model **background:**
  - It is ***behind*** the content (text, image etc)
  - It extends to the border, so it ***includes*** the "padding".
  - It does ***not*** extend past the border (where the "margin" is).
- **background-color:** [colour-rgb] | [colour-hex] | [colour-name] | **transparent**
  - The default background color `transparent` allows the parent element (content / background etc) to show through as the background.
- **background-image**:[url()] | **none**

  Example: `body { background-image: url("tiles.gif");}`
  - If we use a `background-image`, it will be presented over the top of the `background-color`. (For good usability include a background-color with -image)
- **background-position:** top, bottom, left, right, center, [x-% y-%], [x-pos y-pos]

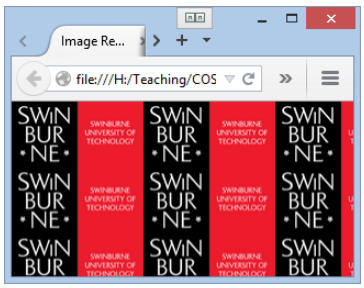
  Note: Percentages will position an image based on center of the image, however constants (eg. 30px) use the top left corner of the image.

  Example: **background-position:** 50% 30px; (50% horizontal (center), 30px vertical (down))

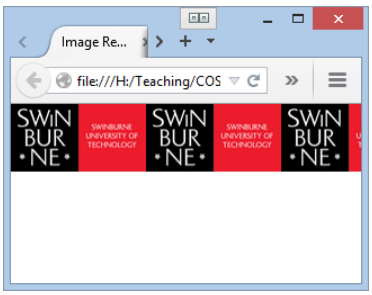
  Default values 0% 0%

CSS: Background Image Repeat

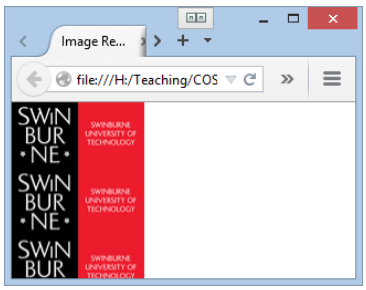
- **background-repeat:** **repeat** | repeat-x | repeat-y | no-repeat



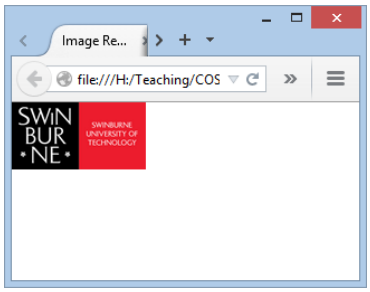
background-repeat: repeat;



background-repeat: repeat-x;



background-repeat: repeat-y;



background-repeat: no-repeat;

Example : Repeats the image along the x (horizontal) axis

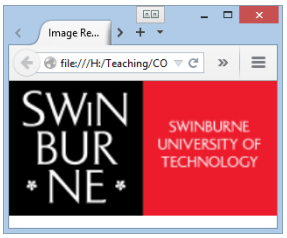
```
body {
        background-image: url("logo.png");
        background-repeat: repeat-x;
}
```

CSS: Background - Some more Properties

- **background-size**: auto | **length** | cover | contain |initial | inherit;

  Example: Length sets the width and height of the background image. The first value sets the width, the second value sets the height. If only one value is given, the second is set to "auto"

```
body{
   background-image: url("logo.png");
   background-size: 120px;        // can also be in % units , ie. scalable
}
```



background-size: cover;

- **background-attachment: scroll** | fixed;

  Example: The background image will stay in the same window location regardless of the browser window scroll.

```
section {
   background-image: url("flowers.gif");
   background-attachment: fixed;
}
```

… and there are many other background properties and values!

Grouped multiple property short-form

- **background:** background-color background-image *background-repeat** *background-attachment** *background-position**

  Example:

```
body {
   background: black url("tile.gif") no-repeat top left;
}
```

***Note**: be aware of **default** and **minimum** values.*

CSS Border

- **Border** surrounds the elements padded content
  - Borders are separated from other elements by the margins.

Grouped multiple property short-form

- **border:**

  border-width border-style border-color

  Example:

```
header { border: 1px dashed #000; }
```

***Note**: be aware of **default** and **minimum** values.*

- **border-style:** (= border-*[all]*-style)

  none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset

- **border-color:** (= border-*[all]*-color)

  [colour-rgb()], [colour-hex], [colour-name]

CSS Border

- We can also specify border grouped properties for individual sides.
  - **border-**[top,right,bottom,left]**:**
    (grouped short form, three properties at once for a side)

    border-width border-style border-color

    Example:

    ```
    h1 {border-bottom: 1px double green; }
    ```
  - **border-**[top,right,bottom,left]**-width:**

    thin, **medium**, thick, [length]
  - **border-**[top,right,bottom,left]**-style:**

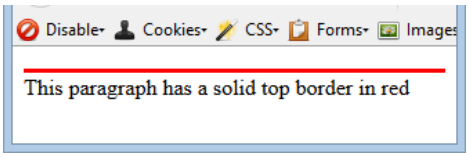
    none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset
  - **border-**[top,right,bottom,left]**-color: none**

    [colour-rgb()], [colour-hex], [colour-name]
    and CSS3 colours

CSS Border

- Example - specifying a top border

```
p {border-top-width: 5px;
   border-top-style: solid;
   border-top-color: red;}
```



OR short form

```
p { border-top: 5px solid red;}
```

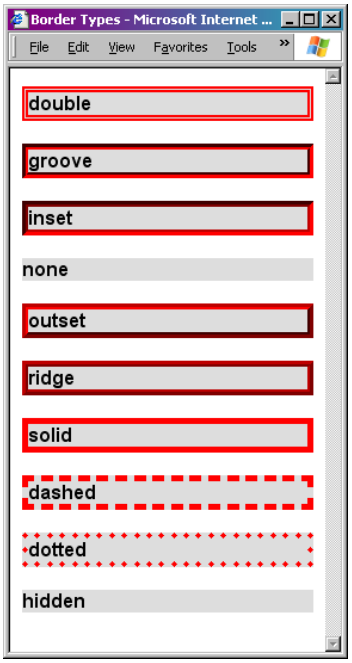
- Example - specifying the borders in the property value

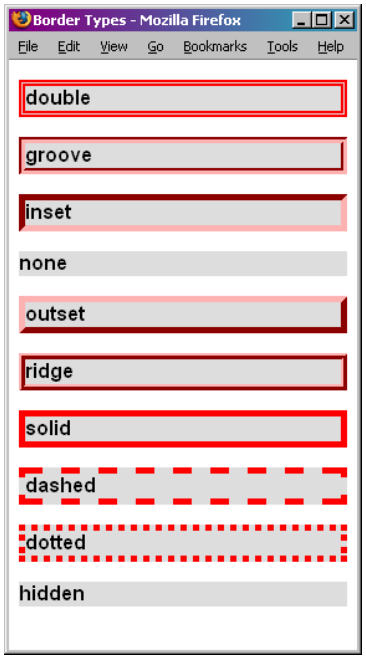
```
h1{ border-width: 3px 4px 2px 1px;
    border-style: dashed solid double dotted;
    border-color: red purple green blue;
```

trouble: top, right, bottom, left

CSS Border -style property values

border-*[top,right,bottom,left]*-style: none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset | initial | inherit





CSS Box Dimensions

- The **`width`** and **`height`** properties can be used to specify the dimensions of block (or "replaced") elements.
  - *They are not "valid" properties for inline elements - unless changed to display:block*
  - If content requires more space than the **`width`** and **`height`** you have specified, the display behaviour is specified by the **`overflow`** property.
  - **`width:`**
    `auto, [length], [%]`
  - **`height:`**
    `auto, [length], [%]`
  - **`max-width, min-width:`**
    `none, [length], [%]`
  - **`max-height, min-height:`**
    `none, [length], [%]`

Note: **`width:`** and **`height:`** (and respective min/max properties) apply to the width and height of the ***content box*** of the element.
The **`padding:`** and **`border:`** of the element are ***outside the specified width and height.***

CSS Margin

- **Margin** allows us to separate elements.
  - Margins do not act as a "fixed buffer" between elements but ensure a ***minimum*** separation. The margins of adjacent elements ***overlap*** and the ***biggest margin*** is the gap that is displayed.

Grouped multiple property short-form:

- **`margin:`**

  `margin-top margin-right margin-bottom margin-left`

  Example:

```
p {margin: 4px 10px 4px 10px; }
```

- Individual margins can be set if needed:
  - **`margin-`***`[top,right,bottom,left]`*`:`**
    `auto, [length], [%]`

    Example:

```
li { margin-top: 4em; }
```


CSS Margin

- The effect of multiple margin values:
  - **Single margin value**, applied to ***all*** sides:

```
p { margin: 4px; }
```

  - **Two margin values:**

```
p { margin: 10em auto; }
```

    - first value (10em) sets the ***top and bottom*** margins
    - second value (auto) applied to the ***left and right*** margins
  - **Four margin values** in clock-wise order (top, right, bottom, left):

```
p { margin: 4px 10px 6px 10px; }
```

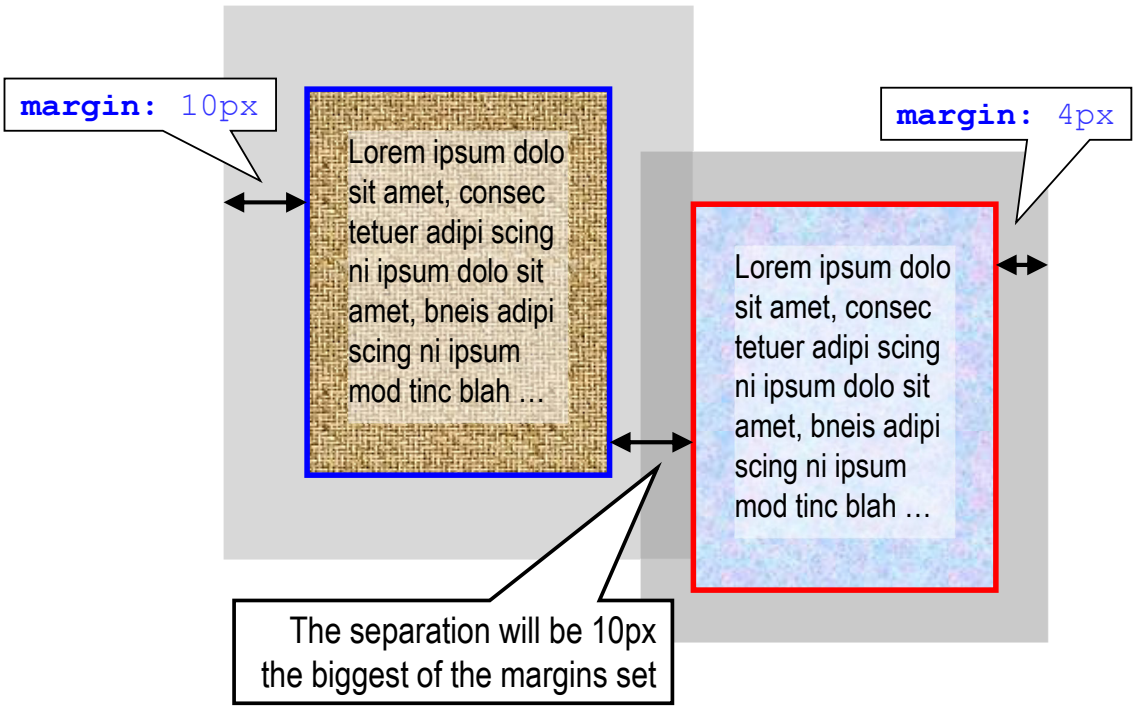
trouble

- We can use the "**`auto`**" margin value to centre an element:

```
table { margin-left: auto; margin-right: auto; }
```

CSS Margin Example

- *Margin is a minimum **separation distance** between elements.*



CSS Padding

- **Padding** is placed between the border and the content.
  (Stops text from being squashed next to the border!)

Grouped multiple property short-form:

- `padding:`
  `padding-top` `padding-right` `padding-bottom` `padding-left`

- Like **`margin`**, we can also use 1,2 or 4 values:

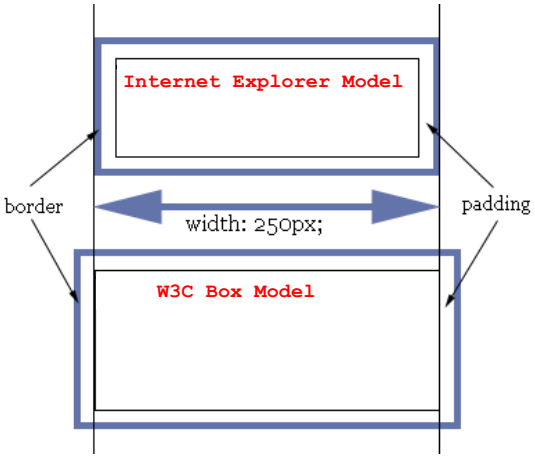
  `padding: 4px;` 4px applied to *all* sides

  `padding: 6px 4px:` 6px *top and bottom*, 4px *left and right*

- Can specify padding for individual sides if we need
  - **`padding-`***`[top,right,bottom,left]`*`:`
    `[length], [%]`

CSS3 - Box Width (and Height)

- *In the W3C CSS2.1 specification, the box width is the **width of the content** - the padding and border is outside*



***Note:** Internet Explorer incorrectly treated the width as outside the border ☹*

CSS3 – introduced
box-sizing: border-box ;
box-sizing: **content-box**;

CSS Properties

- CSS properties define which aspect of the *selected* HTML will be changed or styled
  - Size measurement
  - Colour
  - Typography
    - Fonts
    - Lists
  - **Positioning /Layout**
    - Inline
    - Block Box model
    - **Page Flow**

Page Layout: Design

- **Fluid** */Flexible/Liquid* **layout:**
  one or more elements are set with **relative** units.
  - Layout adapts to the size of the viewport, browser window.
  - Typically related to **width** rather than height
  - Page content "flows" into free areas of the viewport, browser window
- **Fixed layout:** defines exact size of every element in **absolute** units such as pixels.
  - Does not adapt to the size of the browser window
  - Gives precise control over appearance
  - OK for printed page style

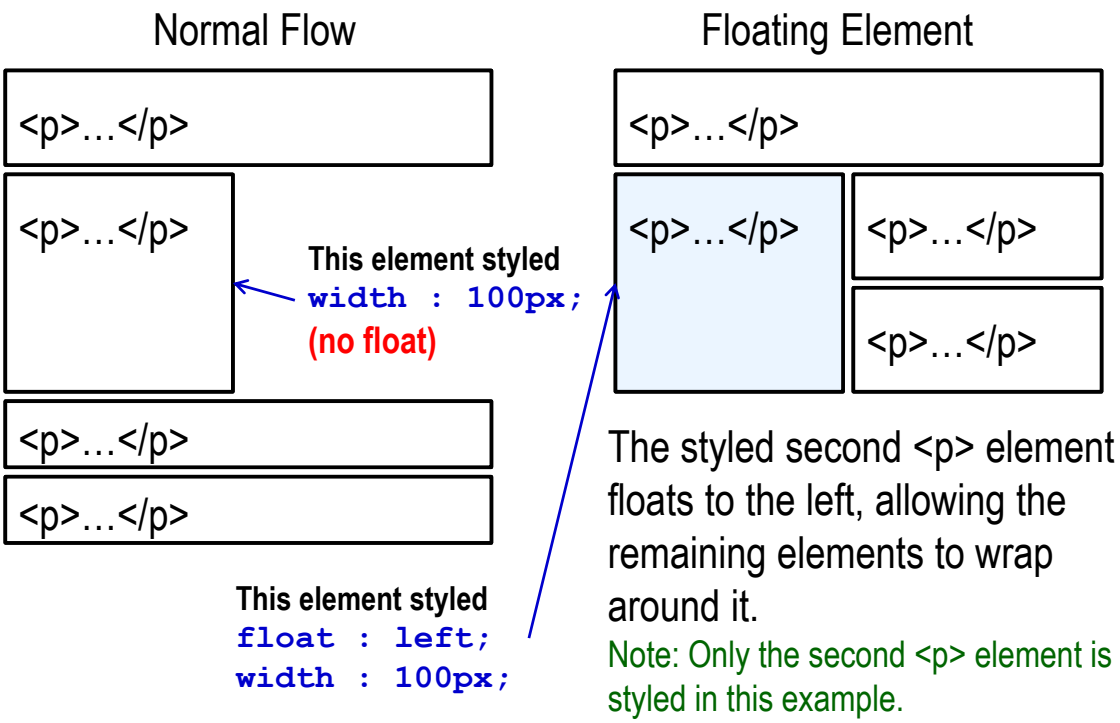
Typically avoid fixed layout

Page Layout: Fluid Flow

- **Normal -** the default browser display of elements, that is one element after the other
  - **Block-level** – vertically from top to bottom
  - **Inline-level** – horizontally from left to right
- **Float** - takes an element out of the normal flow
  - Non-floating elements remain in the normal flow
  - Originally intended to allow text to wrap around images, often used for page layout

CSS: Page Layout: Flow

- float : **none** | left | right;



The styled second <p> element floats to the left, allowing the remaining elements to wrap around it.

Note: Only the second <p> element is styled in this example.

CSS Element Layout

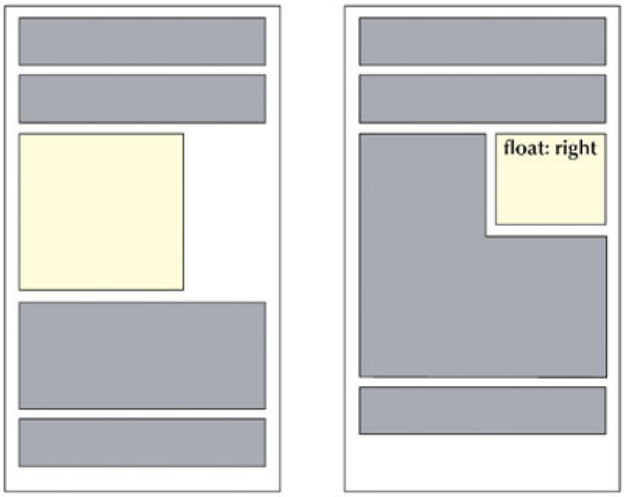
- **float:**
  `left, right, none`
  - Set an element to **float** against the parent border. Other block positions are unaffected, but block contents (eg. text) will flow around the floated element.
- **clear:**
  `left, right, both, none`
  - The **clear** property lets you position elements "clear" from other "floated" elements.

    Example: Make sure that the next "intro" paragraph is clear, both left and right, from any floated images:

```
p.intro {
  clear: both;
}
```

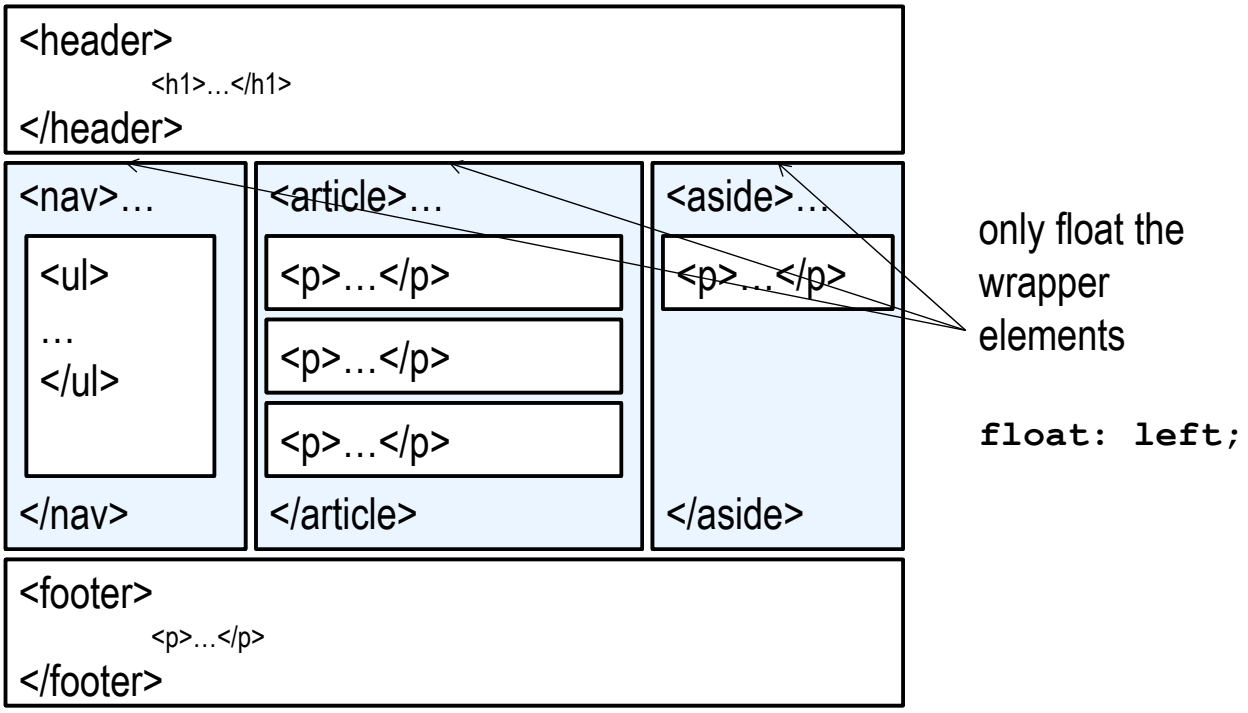
CSS Element Layout Example

- Float example, clear example



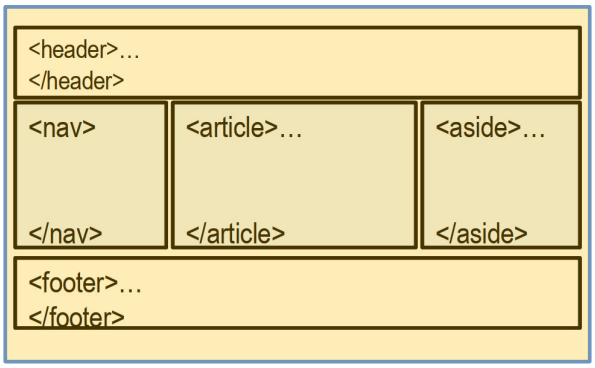
See also CSS Page Layout notes. eg.'float' div blocks into columns
http://css.maxdesign.com.au/floatutorial/

Page Layout: Structural Wrapper Elements

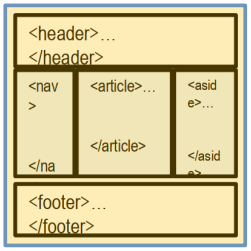


Page Layout: Design – Fluid - *Float*

```
<header >…
 </header>
<nav >…
 </nav>
<article>…
 </article>
<aside>…
 </aside>
<footer>…
 </footer>
```



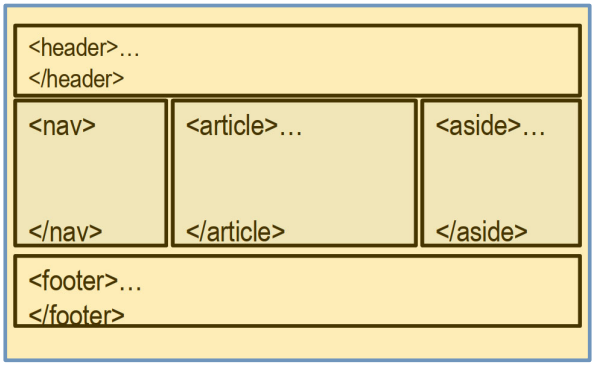
Adapts to the size of the browser window



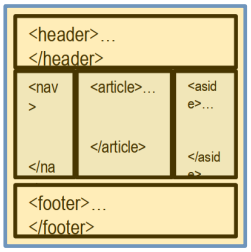

```
header {width:100%;}
nav {width:25%;  float:left;}
article {width:50%; float:left;}
aside {width:20%; float:left;}
footer {width:100%; clear:both;}
```


Page Layout: Design – Fluid – *display:table-cell*

```
<header >…
 </header>
<nav >…
 </nav>
<article>…
 </article>
<aside>…
 </aside>
<footer>…
 </footer>
```



Adapts to the size of the browser window




```
header {width:100%;}
nav, article, aside
{display:table-cell;}
nav {width:25%;}
article {width:50%;}
aside {width:20%;}
footer {width:100%;}
```

CSS Positioning

- In CSS 2.1, a box may be laid out according to three positioning schemes:
  - Normal flow

    Includes block formatting of block-level boxes, inline formatting of inline-level boxes, and relative positioning of block-level and inline-level boxes.
  - Floats

    In the float model, a box is first laid out according to the normal flow, ***then taken out of the flow and shifted*** to the left or right as far as possible. Content may flow along the side of a float.
  - Positioning

    In the positioning model, ***a box is removed from the normal flow entirely*** (it has no impact on later siblings) and is assigned a position with respect to its containing block.

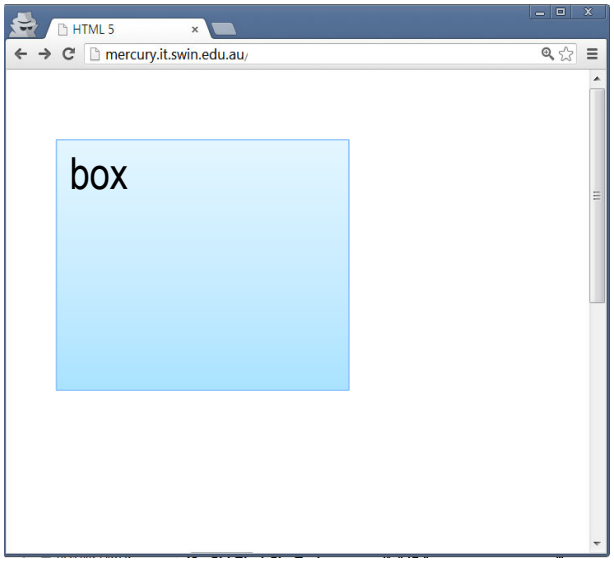
Page Layout: Position, Top and Left

- position: **static** | absolute | fixed | relative;
  - **static** is the default positioning of the elements as they appear in the document flow
  - **relative** positions the element relative to its normal position, (offsetting from static)
  - **absolute** positions the element relative to its first positioned ancestor element
  - **fixed** positions the element relative to the view port or browser window
- Used with top and left property
  - top: **auto** | <value>;
  - left: **auto** | <value>;

Example

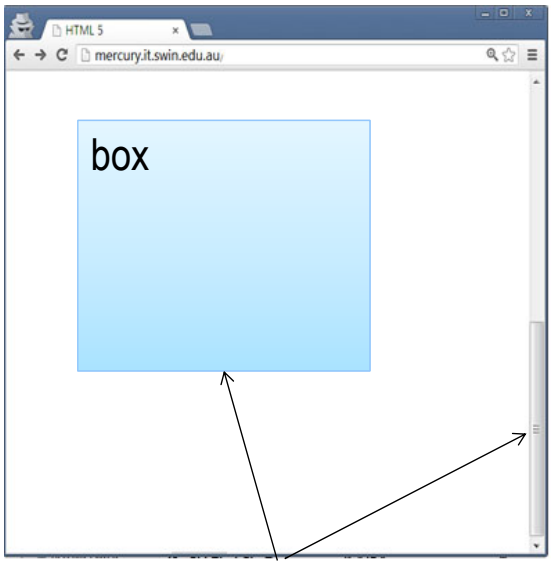
```
width:100px;height:100px;
border:1px solid #black;
background-color:skyblue;
position:absolute;
top:100px;left:100px;
```

- **Avoid `position:` unless really needed**

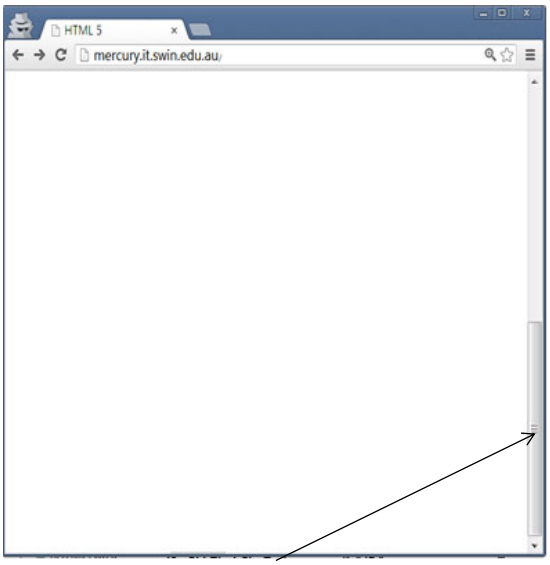


Page Layout: Position, Top and Left

- fixed
- absolute



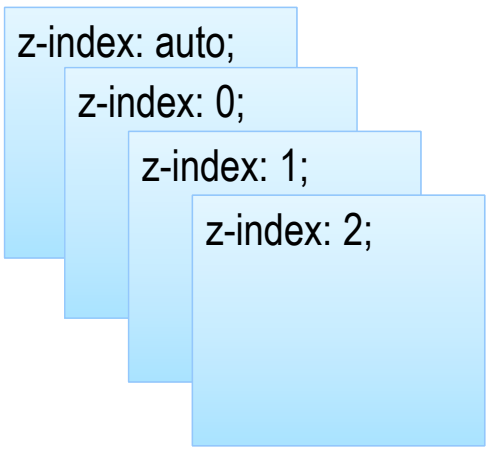
Relative to the window, stays on screen
Even if user scrolls down

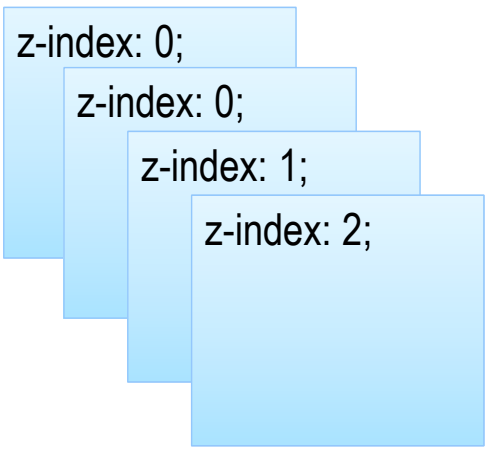


Relative to the page, scrolls with the webpage

Page Layout: z-index

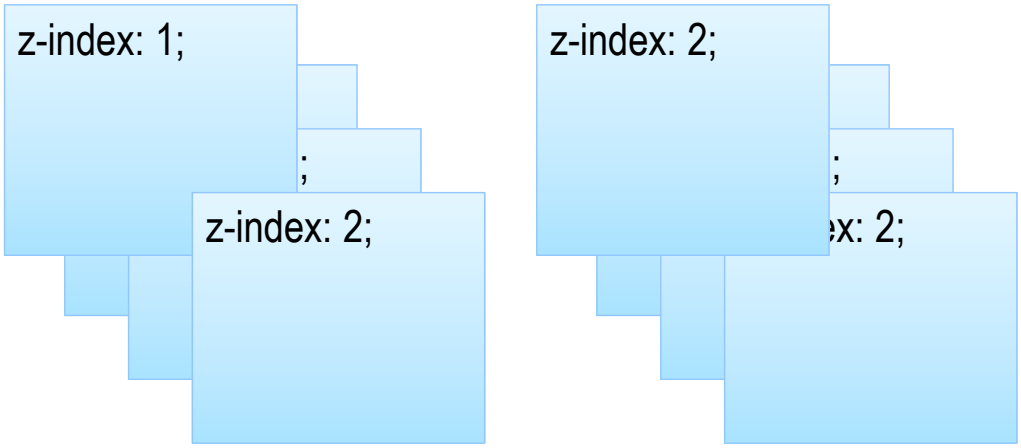
- z-index : **auto** | <number>;
  - Modifies the stacking order of the elements





Page Layout: z-index

- Stacking order of elements with the same z-level value is based on the order in the HTML text



Designing for different devices

- Designing for mobile becoming increasingly important - "**mobile first**"
- "**Responsive design**" is also very important,
  eg. user changes orientation of a mobile device,
  changes screen resolution, changes window size.
  - Web Dev Toolbar | Resize | View Responsive Layouts
- Developed in more detail in
  Mobile Apps development subjects…

CSS: @media

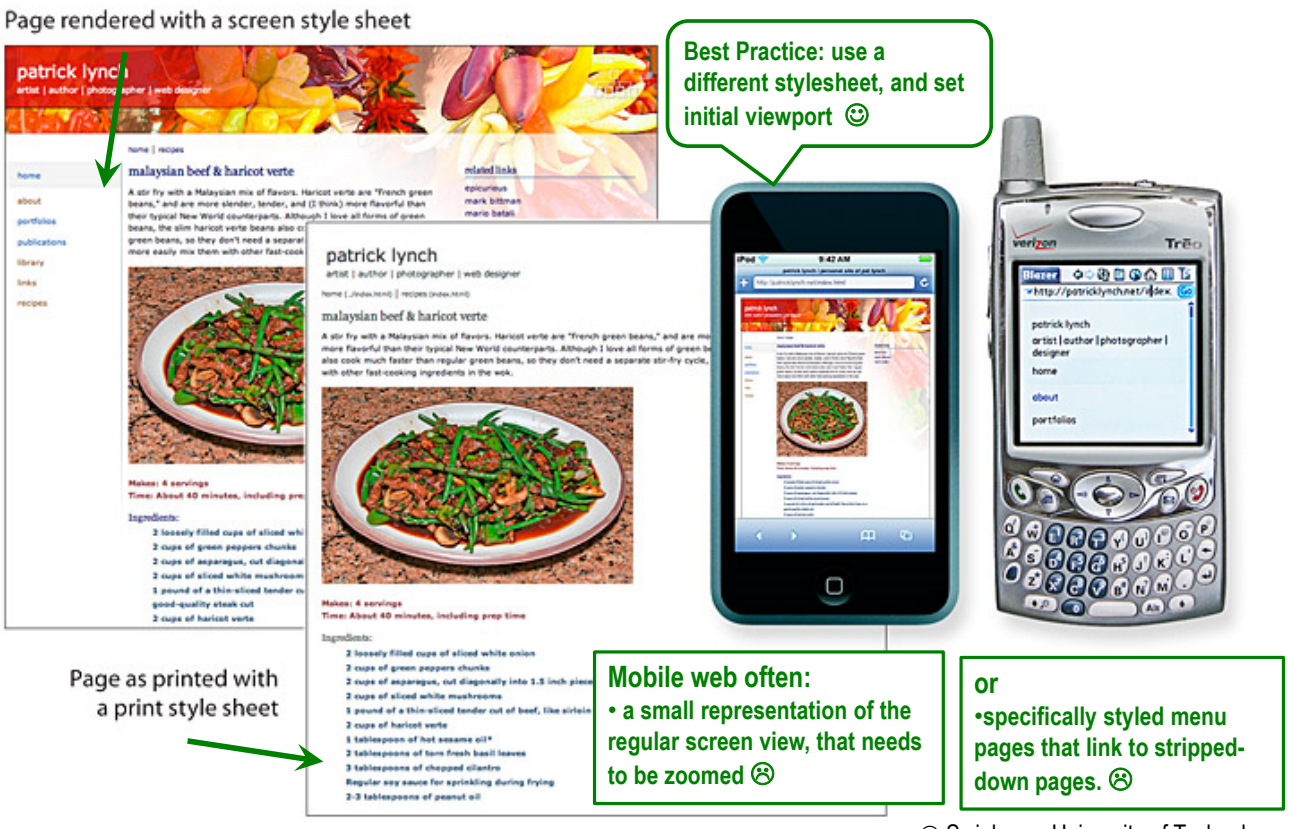
- The `@media` selector is used ***within a single style sheet***, to define style rules for multiple media types.

```
@media screen {
        body {
                font-family: sans-serif; font-size: 18pt;
        }
}

@media print {
        body {
                font-family: serif; font-size: 9pt;
        }
}

@media screen, print {
        body {
                line-height: 150%;
        }
}
```

Example:
View a Wikipedia page In the Browser.
Look at the print preview, with "File/Print Preview".
Note the print style used hides the nav, asides, etc.

http://www.w3schools.com/css/tryit.asp?filename=trycss_media

CSS: Alternate Stylesheets



CSS: Alternate Stylesheets

- The idea of CSS is to be able to have ***different*** style sheets for ***different*** users and ***different*** devices.
- An easy way to offer this is by providing "**alternate**" style sheet links:

```
<link rel="stylesheet"
      href="normal.css" title="normal" />
<link rel="alternate stylesheet"
      href="bigfont.css" title="bigfont" />
<link rel="alternate stylesheet"
      href="aqua.css" title="aqua" />
```

- *The user can select* one of the "**alternate stylesheet**" options.

CSS3: Media Type and Queries

- CSS3 introduced Media Queries, an expansion on the concept of media types in CSS2
  - Media type specify the different style rules
  - Media Queries creates more precise rules
- Both are used for different types of destination media, such as screen, projection, tv, print, embossed, braille, speech, tty, all.
-

```
<link href="mobile_device.css
      rel="stylesheet"
      media="screen and (max-width:480px)" >
```

design breakpoint

CSS3: using the HTML `meta viewport`

- The HTML **`meta viewport`** is widely used to determine the initial "scale" that a web page will be presented in a browser.

```
<meta name="viewport"
content="width=device-width, initial-scale=1 />;
```

- The meta *viewport* is then used with the meta *media attribute* with ***design breakpoints*** to trigger the use of different stylesheets, in response to changes in "window size", "device orientation", "scale", and hence provide ***"responsive web design"***

```
<link href="small.css" rel="stylesheet" media="(max-width:600px)"/>
<link href="large.css" rel="stylesheet" media="(min-width:601px)" />
```

See the optional Responsive Layout Design task in this week's Lab

CSS Frameworks and Pre-processors

- CSS pre-processors
  - e.g. Less, Sass, Stylus, …
  - helps write maintainable, well-structure code
    - e.g. use variables, adds conditional logic
  - reduces the amount of CSS written.
  - good for large-scale user interfaces with many style rules.
- CSS Frameworks/Libraries
  - e.g. Bootstrap, skeleton, Pure CSS, …
  - packaged (and documented) collection of rules
    - e.g. define grid layouts, responsive design, control styling, …
  - usually written using a pre-processor

Do not use in this Subject. We want you to learn the basics.

Be wary, third party CSS libraries need to be loaded on the client, and they change. Avoid using too many CSS tools.

Next Lecture

What's Next?

- Client-side Scripting
- JavaScript