CS472 WEB PROGRAMMING
PAGE LAYOUT:
WHOLE IS GREATER THAN THE SUM OF THE PARTS

Maharishi University of Management -Fairfield, Iowa © 2016

Wholeness Statement

CSS provides different tools for creating a layout. There are a variety of ways to position an element; most of them are based on taking a block level element and placing it in relation to some other block. It is this relationship that becomes the tricky part. The question is always: "This positioned is relative to what?" This illustrates the general principle that individual parts must often be understood in terms of a larger context.

The whole is greater than the sum of its parts.

Main Point Preview

The box model is a description of how every element has a basic width and height, outside of which it has padding, a border, and margin. For inline elements only the left and right margin and padding affect surrounding elements.

The box model is another encapsulation mechanism that allows layout style to be separate from the page content. Life is found in layers.

The CSS Box Model

- For layout purposes, every element is composed of:
  - The actual element's **content**
  - A **border** around the element
  - **padding** between the content and the border (inside)
  - A **margin** between the border and other content (outside)

- width = content width + L/R padding + L/R border + L/R margin
- height = content height + T/B padding + T/B border + T/B margin

- The standard `width` and `height` properties refer ONLY to the content's width and height.

CSS - Box model

Margin
Border
Padding
Height
Width
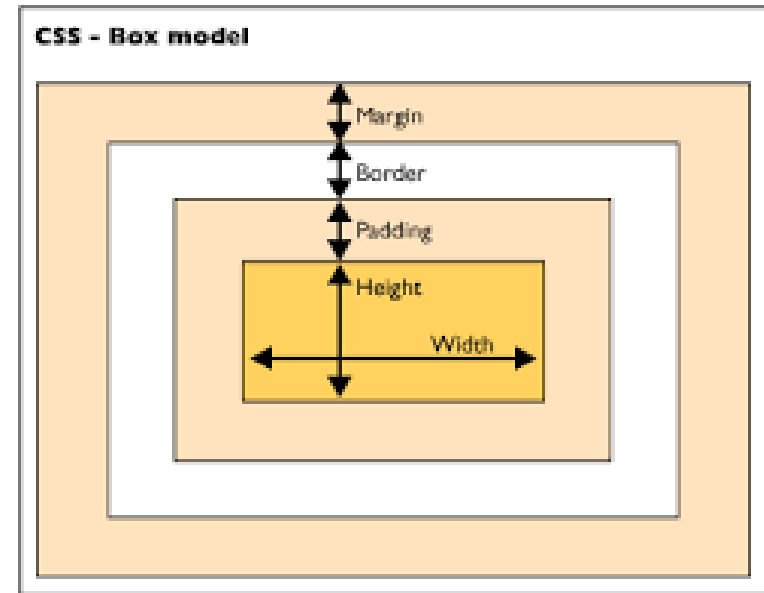
CSS properties for borders

`h2 { `**`border`**`: `**`5px solid red; }`**

This is the best WAP course!

Property	Description
border	thickness/style/color of border on all 4 sides

- **thickness** (specified in px, pt, em, or thin, medium, thick)
- **style** (none, hidden, dotted , dashed , double , groove , inset , outset , ridge , solid)
- **color** (specified as seen previously for text and background colors)

More border properties

Property	Description
border-color, border-width, border-style	specific properties of border on all 4 sides
border-bottom, border-left, border-right, border-top	all properties of border on a particular side
border-bottom-color, border-bottom-style, border-bottom-width, border-left-color, border-left-style, border-left-width, border-right-color, border-right-style, border-right-width, border-top-color, border-top-style, border-top-width	properties of border on a particular side
Complete list of border properties	

Border example 2

```
h2 {
  border-left: thick dotted #CC0088;
  border-bottom-color: rgb(0, 128, 128);
  border-bottom-style: double;
}
```

This is the best WAP course!

- each side's border properties can be set individually
- if you omit some properties, they receive default values (e.g. border-bottom-width above)

Dimensions

• For **Block elements and `img` element only**, set how wide or tall this element, or set the max/min size of this element in given dimension.

width, height, max-width, max-height, min-width, min-height

```
p { width: 350px; background-color: yellow; } h2 { width:
50%; backg                                            }
```

This paragraph uses the first style above.

An h2 heading

➢ Using **`max-width`** instead of **`width`** in this situation will improve the browser's handling of small windows.

  ➢ Max-width will have a smaller width box on smaller viewports versus fixed width will result in extending off screen

The Box Model Caveat

When you set the `width` of an element, the element can actually appear bigger than what you set: the element's `border` and `padding` will stretch out the element beyond the specified width.

```css
.simple {
    width: 500px;
    margin: 20px auto;
}
.fancy {
    width: 500px;
    margin: 20px auto;
    padding: 50px;
    border-width: 10px;
}
```

box-sizing

- `box-sizing: content-box;` - initial and default value. The width and height properties are measured including only the content, but not the padding, border or margin.
- `box-sizing: border-box;` The width and height properties include the content, the padding and border, but not the margin. Note that padding and border will be inside of the box
- Q: Why is Hooray indented?
- responsive design advantage demo

```
.div3 {
width: 300px;
height: 100px;
border: 1px solid blue;
box-sizing: border-box;
}
.div4 {
width: 300px;
height: 100px;
padding: 50px;
border: 1px solid red;
box-sizing: border-box;
}
```
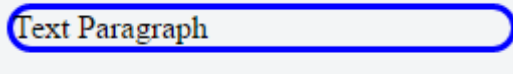
Both divs are the same size now!

Hooray!

Rounded corners `border-radius`

```
p {
  border: 3px solid blue;
  border-radius: 12px;
}
```

Text Paragraph

- Each side's border radius can be set individually, separated by spaces
  - **Four values:** top-left, top-right, bottom-right, bottom-left
  - **Three values:** top-left, top-right and bottom-left, bottom-right
  - **Two values:** top-left and bottom-right, top-right and bottom-left
  - **One value:** all four corners are rounded equally

Padding

- The padding shorthand property sets all the padding properties in one declaration. Padding shares the background color of the element. This property can have from one to four values:

```
padding:10px 5px 15px 20px; /* Top, right, bottom, left */
padding:10px 5px 15px; /* Top, right and left, bottom */
padding:10px 5px; /* Top and bottom, right and left */
padding:10px; /* All four paddings are 10px */
```

- padding-bottom, padding-left, padding-right, padding-top

```
h1 {    padding: 20px; }
h2 {

        padding-left:    200px;

        padding-top:     30px;

}
```

Margin

- Margins are always transparent. This property can have from one to four values:

```css
margin:10px 5px 15px 20px; /* Top, right, bottom, left */
margin:10px 5px 15px; /* Top, right and left, bottom */
margin:10px 5px; /* Top and bottom, right and left */
margin:10px; /* All four margins are 10px */
```

- margin-bottom, margin-left, margin-right, margin-top

```css
h1 {margin: 20px; }
h2 {
    margin-left:    200px;
    margin-top:     30px;
}
```

- See example: lesson3_examples/paddingmargin.html
  - Inspect element in console

Margin Collapse

- Vertical margins on different elements that touch each other (thus have no content, padding, or borders separating them) will collapse, forming a single margin that is equal to the greater of the adjoining margins.
    1. Collapsing Margins Between Adjacent Elements
    2. Collapsing Margins Between Parent and Child Elements

Main Point

The box model is a description of how every element has a basic width and height, outside of which it has padding, a border, and margin. For inline elements only the left and right margin and padding affect surrounding elements.

The box model is another encapsulation mechanism that allows layout style to be separate from the page content. Life is found in layers.

Main Point Preview: Block vs inline elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).   An inline element does not start on a new line and only takes up as much width as necessary. These are the fundamental display types for almost all HTML elements. Understanding these fundamental types is critical to creating effective layouts.

Familiarity with fundamental levels of awareness is critical to successful action.

Details about `block` boxes

- By default **`block`** elements take the entire width space of the page unless we specify.

- To align a **`block`** element at the **`center`** of a horizontal space you must set a **`width`** first, and **`margin: auto;`**

- **`text-align`** does not align **`block`** elements within the page.

Centering a block element: auto margins

```
<p> Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
 incididunt ut la-bore et dolore magna aliqua.
</p>

p {
 border: 2px solid black;
 margin-left: auto;
 margin-right: auto;
 width: 33%;
}
```

- Set the width of a block-level element to prevent it from stretching out to the edges of its container.
  - Set left and right margins to auto to horizontally center that element.
  - Remaining space will be split evenly between the two margins.
- to center inline elements within a block element, use text-align: center;

Details about `inline` boxes

- size properties (width, height, min-width, etc.) are ignored for inline boxes

- margin-top and margin-bottom are ignored, but margin-left and margin-right are not
  - Padding top and bottom ignored

- each inline box's vertical-align property aligns it vertically within its block box

- **text-align** describes how inline content is aligned in its parent block element.
  - does not control the alignment of block elements, only their inline content
 - See lesson3_examples/textalign.html

The`vertical-align` property

- Specifies where an **inline element** should be aligned vertically, with respect to other content on the same line within its block element's box
- Can be top, middle, bottom, baseline (default), sub, super, text-top, text-bottom, or a length value or %.   baseline means aligned with bottom of surrounding characters

```
img {
  vertical-align: baseline;
}
img {
  vertical-align: middle;
}
```

- See common error example:
          lesson3_examples/verticalaligin.html
- image is vertically aligned to the baseline of the paragraph, which isn't the same as the bottom
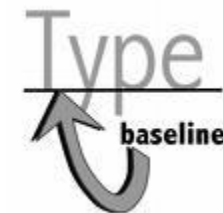
(review) *display* *property*: block vs inline

- The default value of display **property** for most elements is usually **block** or **inline**.
- **Block: div** is the standard block-level element. A block-level element starts on a new line and stretches out to the left and right as far as it can. Other common block-level elements are **p** and **form**, and new in HTML5 are **header**, **footer**, **section**, and more.
- **Inline: span** is the standard inline element. An inline element can wrap some text inside a paragraph **** like this **** without disrupting the flow of that paragraph. The **a** element is the most common inline element, since you use them for links.
- **None**:   Another common display value is **none**. Some specialized elements such as **script** use this as their default. It is commonly used with JavaScript to hide and show elements without really deleting and recreating them.
- **inline-block** elements are like **inline** elements but they can have a width and height.
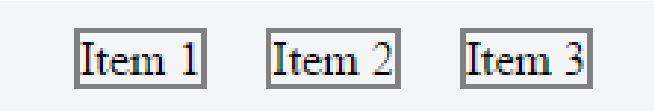- Flex and grid: new options for responsive layouts

Displaying `block` elements as `inline`

- Lists and other **block** elements can be displayed **inline**, flow left-to-right on same line.

  Note: Width will be determined by content (while block elements are 100% of page width)
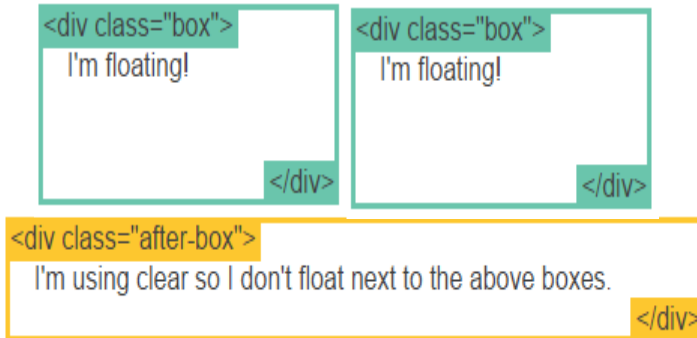


```html
<ul id="topmenu">
 <li>Item 1</li>
 <li>Item 2</li>
 <li>Item 3</li>
 </ul>
```

```css
#topmenu li {
 display: inline;
 border: 2px solid gray;
 margin-right: 1em;
 list-style-type: none;
}
```

Boxes (Photo Gallery)

The Hard Way (using `float`)
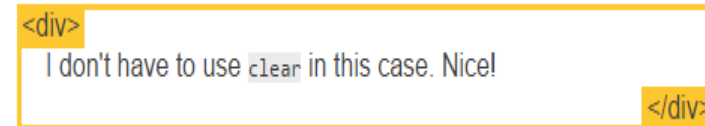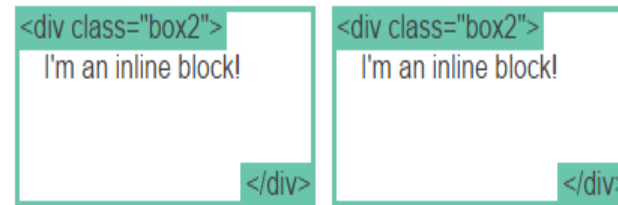
The Easy Way (using `inline-block`)


```css
.box {
 float: left;
 width: 200px;
 height: 100px;
 margin: 1em;
}
.after-box {
 clear: left;
}
```

```css
.box2 {
 display: inline-block;
 width: 200px;
 height: 100px;
 margin: 1em;
}
```

nav links another possible use case for inline-block display

Visibility vs Display

- Setting **display** to **none** will render the page as though the element does not exist.

- **visibility**: **hidden**; will hide the element, but the element will still take up the space it would if it was fully visible
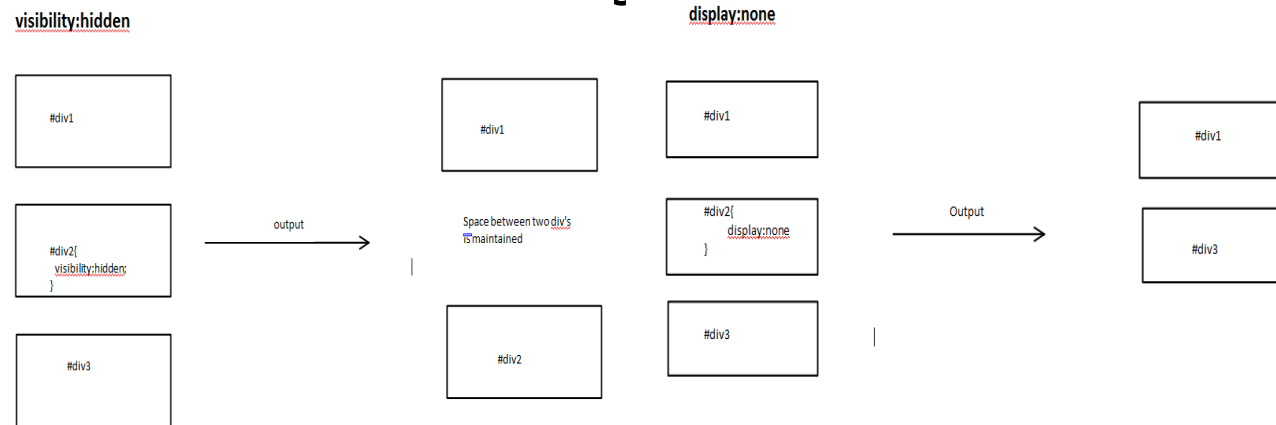
display: none vs visibility: hidden

Opacity

- The **opacity** property sets the opacity level for an element.
- Value ranges from 1.0 (opaque) to 0.0 (transparent)
- [Example usage](#) with :hover

```
div {
 opacity: 0.5;
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Phasellus imperdiet, nulla et dictum interdum, nisi lorem
egestas odio, vitae scelerisque enim ligula venenatis dolor.

Main Point: Block vs inline elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).   An inline element does not start on a new line and only takes up as much width as necessary. These are the fundamental display types for almost all HTML elements. Understanding these fundamental types is critical to creating effective layouts.

Familiarity with fundamental levels of awareness is critical to successful action.

Main Point preview

Static position flows box elements from top to bottom, and inline elements from left to right. **Relative** position keeps the space in the original flow but displays the element at an offset. **Absolute** position takes the element out of the flow and places it relative to the "containing element". **Fixed** position takes the element out of the flow and places it relative to the viewport.

Layouts require understanding how parts fit into a larger whole. The whole is greater than the sum of its parts.

The `position` property

Property	Meaning	Values
position	Location of element on page	**static**: default position **relative**: offset from its normal static position **absolute**: at a particular offset within its containing element **fixed**: at a fixed location within the browser window
top, bottom, left, right	Offsets of element's edges	A size in px, pt, em or %

`position:static;`

- **static** is the default position value for all elements.
- An element with **position: static;** is not positioned in any special way.
- A static element is said to be not positioned and an element with its position set to anything else is said to be positioned.

postion:relative;

- Set the location of an element to an offset from its normal static position.
- **relative** behaves the same as **static** unless you add some extra properties. Setting the **top**, **right**, **bottom**, and **left** properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element. **relative** element stays in its place!

```
<p> This example has <span id="lifted">some text</span> with a relative
position. </p>


#lifted {
 position: relative;
 left: 2em;
 top: 1em;
 border: 2px solid black;
}
```

See example: lesson3_examples/position-relative.html

This example has ____ with a relative position.
some text

position: absolute;

- Elements that are positioned relatively are still considered to be in the normal flow of elements in the document.

- In contrast, an element that is positioned absolutely is taken out of the flow and thus takes up no space when placing other elements.

- The absolutely positioned element is positioned relative to *nearest positioned ancestor (non-static)*. If a positioned ancestor doesn't exist, the initial container is used.

See example:

lesson3_examples/position-absolute.html

lesson3_examples/position-absoluteblock.html

```
#menubar{
 width: 100px;
 height: 50px;
 position: absolute;
 top: 20px;
 left: 50px;
}
```

position: fixed;

- Fixed positioning is similar to absolute positioning, with the exception that the element's containing block is the viewport. This is often used to create a floating element that stays in the same position even after scrolling the page.

```
#one {
 position: fixed;
 top: 80px;
 left: 10px;
}
```

See example:
lesson3_examples/position-fixed.html

- A fixed element does not leave a gap in the page where it would normally have been located.
- A fixed element loses its space in the flow
- A fixed element does not move when you scroll (stays in place)

Overlapping Elements

- When elements are positioned, they can overlap other elements.

- The z-index property specifies the stack order of an element

- An element can have a positive or negative stack order:

Main Point

Static position flows box elements from top to bottom, and inline elements from left to right. **Relative** position keeps the space in the original flow but displays the element at an offset. **Absolute** position takes the element out of the flow and places it relative to the "containing element". **Fixed** position takes the element out of the flow and places it relative to the viewport.

Layouts require understanding how parts fit into a larger whole. The whole is greater than the sum of its parts.

Main Point preview

- The CSS float property makes its element move to right or left side of the containing box. The clear property moves its element downwards if there is a floating element on the specified side. Float is a convenient way to have text wrap around an element or make something appear on the right or left side

- *Do less and accomplish more.*

float

Float has several important uses for layout

- One is for wrapping text around images.
- Another is to position elements on the left or right or center, and possibly columns (being replaced by Flexbox and grid)

```
img {
    float: right;
    margin: 0 0 1em 1em;
}
```

- When we float a `div` element it will comply to `width` and `height` properties rather than behaving like a block element (does not take whole width)
- A floating element is removed from normal document flow.
- Underlying text wraps around it as necessary.
- See example: lesson3_examples/floatrightimage.html

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis.

Common float bug: missing width

- often floating block elements must have a width property value
  - if no width is specified, the floating element will size to fit its content. Large content may occupy 100% of the page width, so no content can wrap around it
  - See example: lesson3_examples/floatingwithoutwidth.html

The `clear` property

(Stop the float I want to get off!)

```
img.hoveringicon { float: left; margin-right: 1em; }
h2 { clear: left; background-color: yellow; }
p { background-color: fuchsia; }
```

Starhome Sprinter is a Flash animated Internet cartoon. It mixes surreal humour with references to 1980s and 1990s pop culture, notably video games, classic television and popular music.

**My Starhome Sprinter Fan Site**

See example: lesson3_examples/clear.html

Property	Meaning	Values
clear	Whether to move this element below any prior floating elements in the document	left, right, both, none(default)

Common error: container too short

- If you place a tall *floating element* inside a block element without much other content, the floating element may hang down past the bottom edge of the block element that contains it.
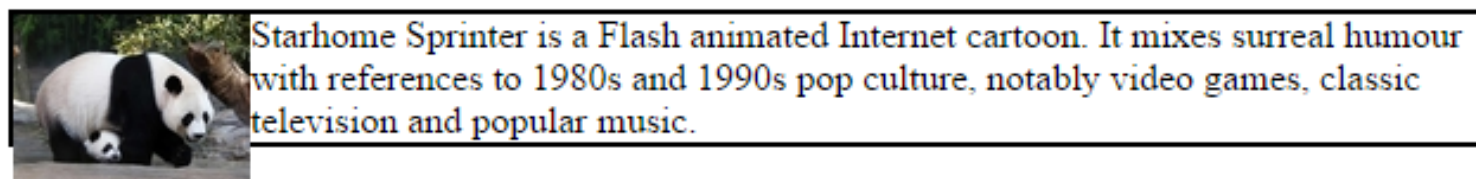
- See example: lesson3_examples/commonerrorcontenttooshort.html

The `overflow` property

overflow: hidden

2 scenarios for use of overflow

- *Floating elements* that are too tall for containing block
  - Previous slide
  - lesson3_examples/overflow.html
- Blocks of fixed height that have more content than their space allows
  - https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_visible

Property	Meaning	Values
overflow	Action to take if element's content is larger than the element itself	visible(default), hidden, scroll, auto

Multi-column layouts

- When more than one element floats in the same direction, they stack horizontally

```css
div, p { border: 2px solid black; }
.column { float: right; width: 25%; }
```

```html
<div class="column"> Lorem ipsum dolor sit amet, consectetuer
 adipiscing elit. Integer pretium dui sit amet felis. </div>
<div class="column"> Integer sit amet diam. Phasellus ultrices
 viverra velit. </div>
<div class="column"> Beware the Jabberwock, my son! The jaws
 that bite, the claws that catch! </div>
```

See example: lesson3_examples/multicolumn-float.html

Multi-column

```
#columns {
  column-count: 3;
  column-gap: 40px;
  column-rule: 2px dotted gray;}
```

lesson3_examples/multicolumn-count.html

browser splits columns rather than split into smaller divs ourselves.

Property	Description	Values
column-count	Number of columns to use	an integer
column-fill	How to choose columns' size	balance(default), auto
column-gap	Space between columns	a size (px, pt, %, em)
column-rule	Vertical line between columns	a width, style and color
column-span	Lets element span many columns	1(default) or all
column-width	width of each column	a size (px, pt, %, em)

Block elements behavior with `position/float`

- One thing to remember, that once a block element is positioned as **`fixed`** or **`absolute`**, it will ONLY occupy the space of its content rather than taking the whole width space.

- Same thing applies for block elements with **`float`**.
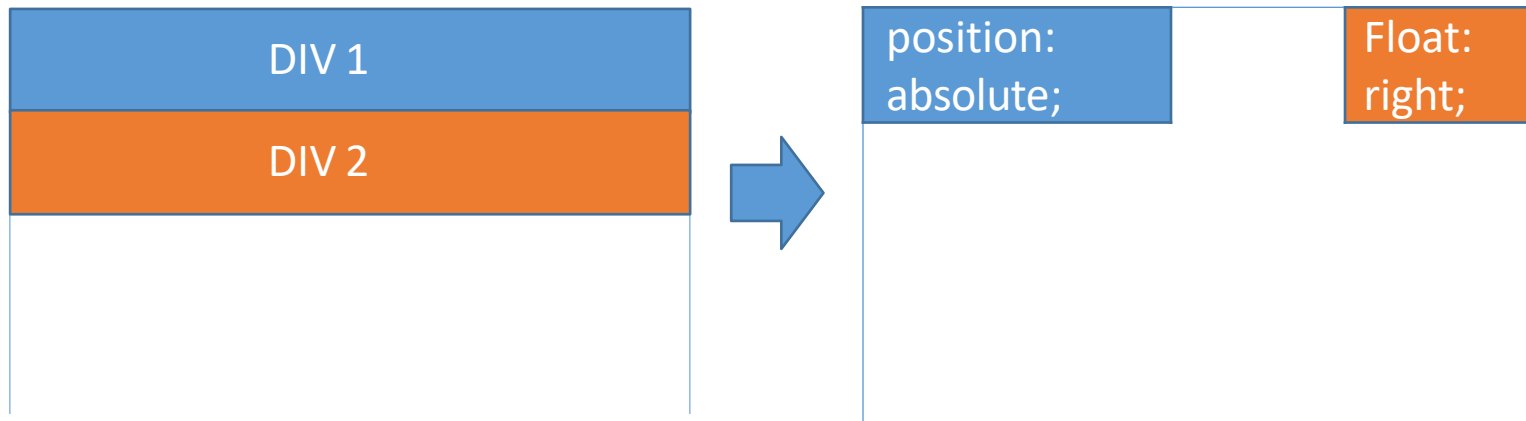
Alignment vs. float vs. position

1. if possible, lay out an element by aligning its content
   - horizontal alignment: text-align
     - <u>set this on a block element; it aligns the content (text and inline elements) within</u> it (not the block element itself)
     - E.g., see lesson3_examples/textalign.html example
     - Might need to make block elements have display: inline
   - vertical alignment: vertical-align
     - set this on an inline element, and it aligns it vertically within its containing element
2. if alignment won't work, try floating the element
3. if floating won't work, try positioning the element
   - absolute/fixed positioning are a last resort and should not be overused
4. OR, use flexbox or grid instead of float and positioning

Main Point

- The CSS float property makes its element move to right or left side of the containing box. The clear property moves its element downwards if there is a floating element on the specified side. Float is a convenient way to have text wrap around an element or make something appear on the right or left side

- *Do less and accomplish more.*

Main Point Preview Responsive Design

- **Responsive design** is the strategy of making a site that responds to the browser and device width. Responsive design utilizes media queries to determine the available display area and new CSS techniques such as flexbox and grid to make designs more flexible.

- *The unified field is the source of all possibilities and when we think from this level our actions are spontaneously responsive to whatever situation we encounter.*

Responsiveness guidelines

- (Mobile) users scroll websites vertically not horizontally!
- if forced to scroll horizontally, or zoom out it results in a poor user experience.
- Some additional rules to follow:
  - 1. Do NOT use large fixed width elements
  - 2. Do NOT let the content rely on a fixed viewport width to render well
  - 3. Use CSS media queries to apply different styling for small and large screens
  - https://www.w3schools.com/css/css_rwd_viewport.asp
  - https://www.quirksmode.org/blog/archives/2010/09/combining_meta.html
  - https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag
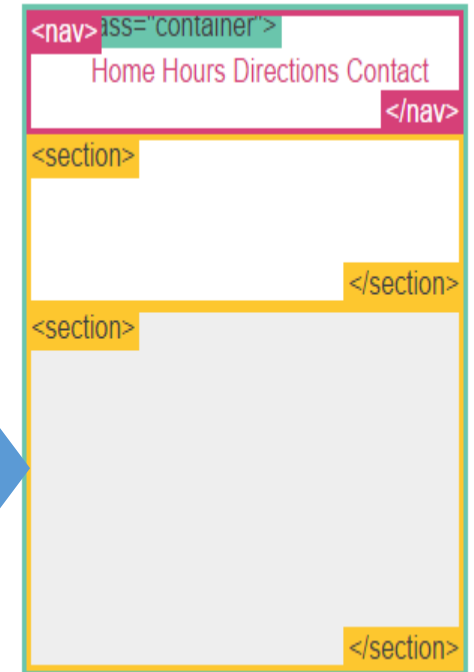
`Media` queries

- **Media queries specify a screen size for which a set of CSS rules apply**

```
@media screen and (min-width:600px) {
  nav { float: left; width: 25%; }
  section { margin-left: 25%; }
}
@media screen and (max-width:599px) {
  nav li { display: inline; }
}
```


- See lesson3_examples\mediaquery.html
- example similar to figure

Design for Mobile First

- Mobile First means designing for mobile before designing for desktop or any other device

- Instead of changing styles when the width gets *smaller* than 768px, we should change the design when the width gets *larger* than 768px.

- @media only screen and (max-width: 600px) /* applies to any screen 600px or smaller -- avoid this */

- @media only screen and (min-width: 600px) /* applies to any screen 600px or larger -- favor this */

meta viewport

- Pages optimized to display well on mobile devices should include meta_viewport in head
  - <meta name=viewport content="width=device-width, initial-scale=1">
  - gives browser instructions to control the page's dimensions and scaling

- Some mobile browsers, notably Safari iPhone, have a default layout viewport of around 850 to 1000 pixels — much larger than the device width.
  - Why? In order to accommodate desktop sites whose developers did not test on mobile.
  - Such sites usually stretch to roughly that width

- If you use media queries without a meta viewport tag in that situation they would misfire horribly.
  - Your code would conclude the browser is larger than 600 pixels, and apply the desktop styles instead of the mobile ones. Adding the meta viewport tag solves that problem

  - View from tablet or phone
    - Without viewport tag
    - With viewport tag

Flexbox Layout

➢ responsive layout without float or positioning

➢ set *display* property to *flex* on the containing element

 ➢ direct child elements are automatically flexible items

 ➢ simple example

➢display: flex;

flex container properties

➢ <u>flex direction</u>: column or row

➢ <u>flex wrap</u>: wrap or nowrap

➢ <u>center vertically and horizontally</u> via justify-content and align-items

➢ <u>Responsive image grid</u>

➢ <u>Responsive website</u>

<u>display</u>	Specifies the type of box used for an HTML element
<u>flex-direction</u>	Specifies the direction of the flexible items inside a flex container
<u>justify-content</u>	Horizontally aligns the flex items when the items do not use all available space on the main-axis
<u>align-items</u>	Vertically aligns the flex items when the items do not use all available space on the cross-axis
<u>flex-wrap</u>	Specifies whether the flex items should wrap or not, if there is not enough room for them on one flex line
<u>align-content</u>	Modifies the behavior of the flex-wrap property. It is similar to align-items, but instead of aligning flex items, it aligns flex lines
<u>flex-flow</u>	A shorthand property for flex-direction and flex-wrap

Grid Layout

- CSS Grid offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

- A Grid Layout must have a parent element with the display property set to grid

- Direct child element(s) of the grid container automatically becomes grid items.

Grid vs Flexbox vs Multi-column

- **Grid is designed to be used *with* flexbox, not instead of it**
- Flexbox is for one dimensional layout (row or column).
- CSS grid is for two dimensional page layout.
- Multi-column best for columns with flow of content across the columns
    - E.g., flowing body of text across several columns

grid-template-columns property

- grid-template-columns: number and width of columns

```
.grid-container {
        display: grid;
         grid-template-columns: 80px 200px auto 40px;
        }
```

- grid-template-rows: height of rows

```
grid-template-rows: 80px 200px;
```

Child Elements (Items)

- A grid *container* contains grid *items*.
- Default: one grid item for each column in each row
  - can also span multiple columns and/or rows

grid-column property

➢ grid-column: which columns an item covers

starts on line 1 and ends on line 5:
.item1 {
  grid-column: 1 / 5; /* n columns have n+1 lines , from line 1 at left side to line n+1 on right */
}

starts on column 2 and spans 3 columns:
.item2 {
  grid-column: 2 / span 3;
}

grid-row property

- grid-row: which rows an item covers

- start on row-line 1 and end on row-line 4:
.item1 {
  grid-row: 1 / 4;
}

- start on row 2 and span 2 rows:

.item2 {
  grid-row: 2 / span 2;
}

grid-template-areas property

grid-template-areas: name items and use the names to layout columns and rows

.item1 { grid-area: header; } /* assign names */
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }

.grid-container {
 grid-template-areas:
   'header header header header header header'
   'menu main main main right right'
   'menu footer footer footer footer footer';
}

good example and exercise https://www.w3schools.com/css/css_templates.asp

CSS Frameworks

Because CSS layout is so tricky, there are CSS frameworks out there to help make it easier. Here are a few if you want to check them out. Using a framework is only a good idea if the framework really does what you need your site to do. They're no replacement for knowing how CSS works.

➢ Blueprint

➢ Bootstrap

➢ Foundation

➢ SemanticUI
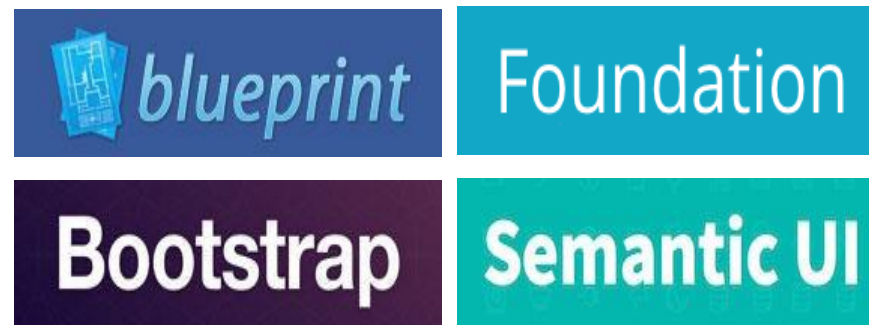
Bootstrap grid system has responsive breakpoints

> [create a](#) row (<div class="row">).
> > add columns (tags with appropriate .col-*-* classes)
> > first star (*) represents the responsiveness: sm, md, lg or xl
> > second star represents a number, which should add up to 12 for each row

```
<div class="row">
  <div class="col-sm-4">.col-sm-4</div>
  <div class="col-sm-8">.col-sm-8</div>
</div>
```

- col-lg, stack when the width is < 1200px.
- col-md, stack when the width is < 992px.
- col-sm, stack when the width is < 768px.
- col-xs, then the columns will never stack.

Bootstrap examples

➢ [Bootstrap table]

    ➢ app specific modification to adjust the color of the striped rows

    ➢ important to understand css and not just blindly use bootstrap

➢ [carousel]

➢ [basic bootstrap4 website]

    ➢ header, nav, 2 col with links vs content…, footer

Main Point Responsive Design

- **Responsive Design** is the strategy of making a site that responds to the browser and device width. Responsive design utilizes media queries to determine the available display area and new CSS techniques such as flexbox and grid to make designs more flexible.

- *The unified field is the source of all possibilities and when we think from this level our actions are spontaneously responsive to whatever situation we encounter.*

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Page Layout: Whole Is Greater than the Sum of the Parts

1. You can use floats and positioning to change where elements are displayed.
2. Modern web apps use responsive design principles including media queries, viewport, Flexbox, grid, and frameworks such as Bootstrap

_____

3. **Transcendental consciousness** is the experience of pure wholeness.
4. **Impulses within the Transcendental field**: At quiet levels of awareness thoughts are fully supported by the wholeness of pure consciousness.
5. **Wholeness moving within itself:** In Unity Consciousness, one appreciates all parts in terms of their ultimate reality in wholeness.