

# CS472 WAP

## Lecture 3: Layout

---

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

## Maharishi International University - Fairfield, Iowa © 2024



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi International University.

# 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. 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.*

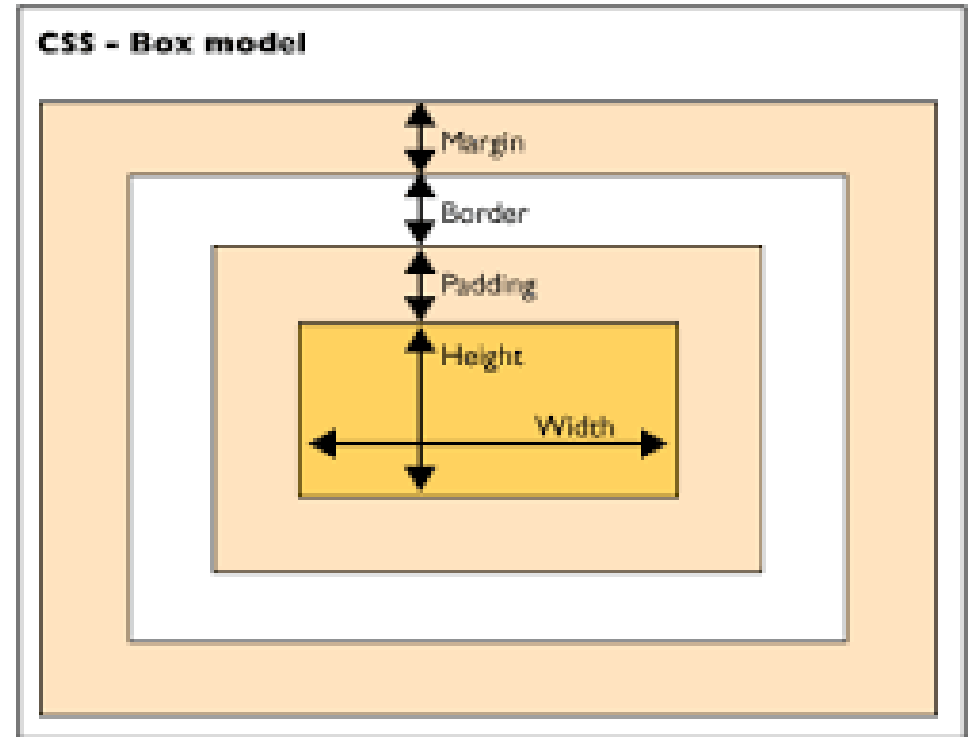
# Main Point Preview

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 an abstraction that allows many similar elements to be treated by a common set of rules. More abstract levels of awareness are more powerful.*

# The CSS Box Model

- For layout purposes, every element is composed of:
  - **content**
  - **border**
  - **padding**
  - **margin**
- $\text{width} = \text{content width} + \text{L/R padding} + \text{L/R border} + \text{L/R margin}$
- $\text{height} = \text{content height} + \text{T/B padding} + \text{T/B border} + \text{T/B margin}$
- The standard `width` and `height` properties refer ONLY to the content's width and height.



# 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
<a href="#">Complete list of border properties</a>	



# 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)

## Rounded corners `border-radius`



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

A visual representation of the CSS rule, showing a light gray rectangular box with a thick blue border and rounded corners. Inside the box, the text 'Text Paragraph' is written in a serif font.

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

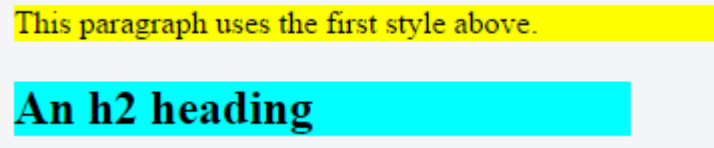
# Dimensions

- 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%; background-color: aqua; }
```



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

# 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:

```
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

```
h1 { margin: 20px; }
```

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

# 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

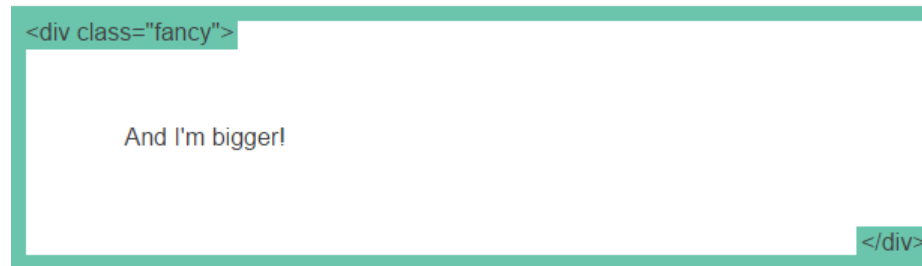
# The Box Model Caveat

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.

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



The diagram shows a rectangular box with a thin green border. Inside the box, the text "I'm smaller..." is centered. The box is labeled with the HTML code `<div class="simple">` at the top left and `</div>` at the bottom right.



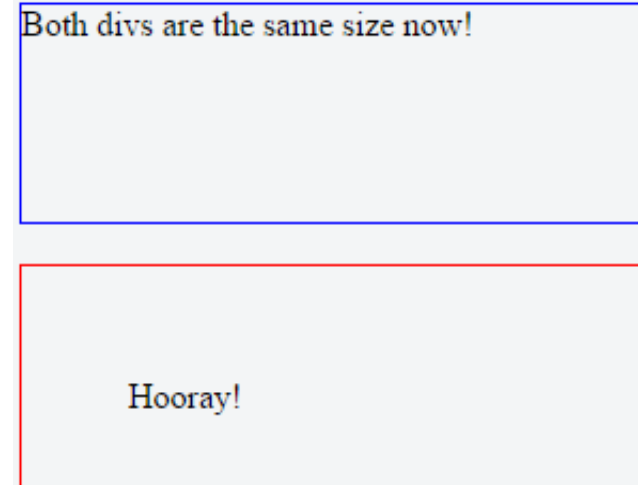
The diagram shows a rectangular box with a thick green border. Inside the box, the text "And I'm bigger!" is centered. The box is labeled with the HTML code `<div class="fancy">` at the top left and `</div>` at the bottom right.

# 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

```
.div3 {  
  width: 300px; height: 100px;  
  border: 1px solid blue;  
  box-sizing: border-box;  
}  
.div4 {  
  width: 300px; height: 100px;  
  border: 1px solid red; padding: 50px;  
  box-sizing: border-box;  
}
```





# 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 an abstraction that allows many similar elements to be treated by a common set of rules. More abstract levels of awareness are more powerful.*

# 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;  
  width: 33%;  
  margin-left: auto;  
  margin-right: auto;  
}
```

- 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

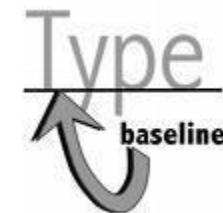
# The `vertical-align` property

- 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 non-hanging letters

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

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

- 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 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 standard inline element. An inline element can wrap some text inside a paragraph **<span>** like this **</span>** 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

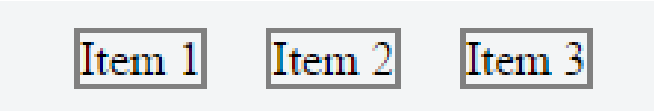
# Displaying **block** elements as **inline**



- 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)*

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



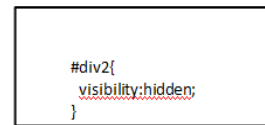
```
#topmenu li {
  display: inline;
  border: 2px solid gray;
  margin-right: 1em;
  list-style-type: none;
}
```



# 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.

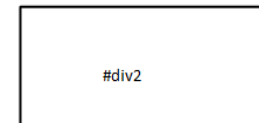
visibility:hidden



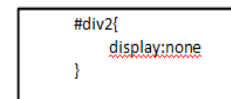
output →



Space between two div's  
is maintained



display:none



Output →



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	<b>static</b> : default position <b>relative</b> : offset from its normal static position <b>absolute</b> : at a particular offset within its containing element <b>fixed</b> : 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.

# position: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:20px;  
  top:10px;  
  border: 2px solid black;  
}
```

This example has some text with a relative position.

# position: absolute;



- position relative still in the normal flow of elements
- position absolute is out of flow and takes up no space
  - positioned relative to *nearest positioned ancestor*
  - If positioned ancestor doesn't exist, initial container is used.
- common idiom
  - use enclosing div with relative position, but no left, top, right or bottom values
  - outer div stays at normal static position
  - point of reference for absolutely positioned elements inside



# position: fixed;



- fixed position like absolute position, except **containing block is viewport**
- often used to create floating element in same position even after scrolling
- does not leave a gap in the page where it would normally have been located.
- loses its space in the flow
- 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 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.w3schools.com/css/css_rwd_viewport.asp)
  - [https://www.quirksmode.org/blog/archives/2010/09/combining\\_meta.html](https://www.quirksmode.org/blog/archives/2010/09/combining_meta.html)
  - [https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport\\_meta\\_tag](https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag)

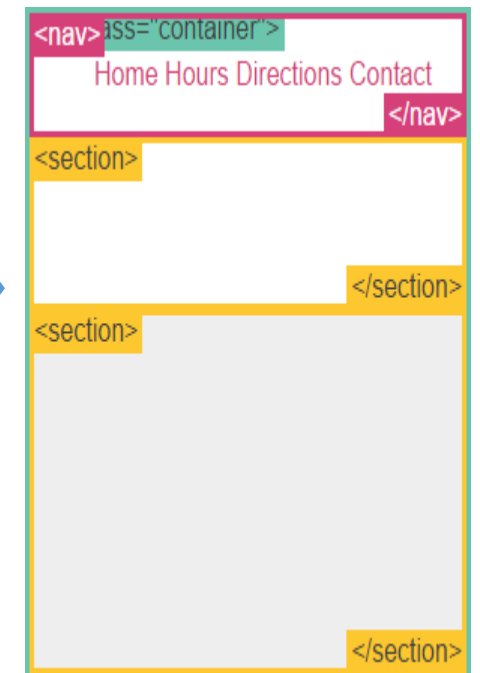
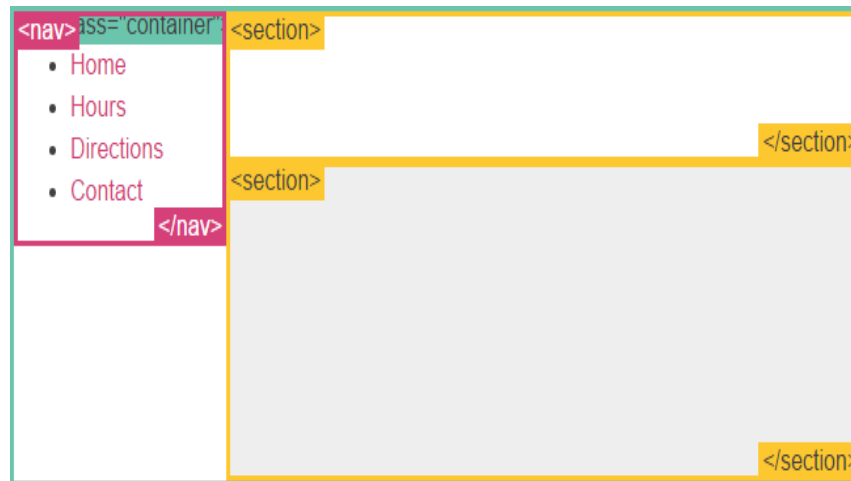
# Media queries



- 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; }  
}
```



- [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 \*/

# Flexbox Layout

- responsive layout without float or positioning
- set *display* property to *flex* on the containing element
- direct child elements are automatically flexible items

`display: flex;`

[simple example](#)



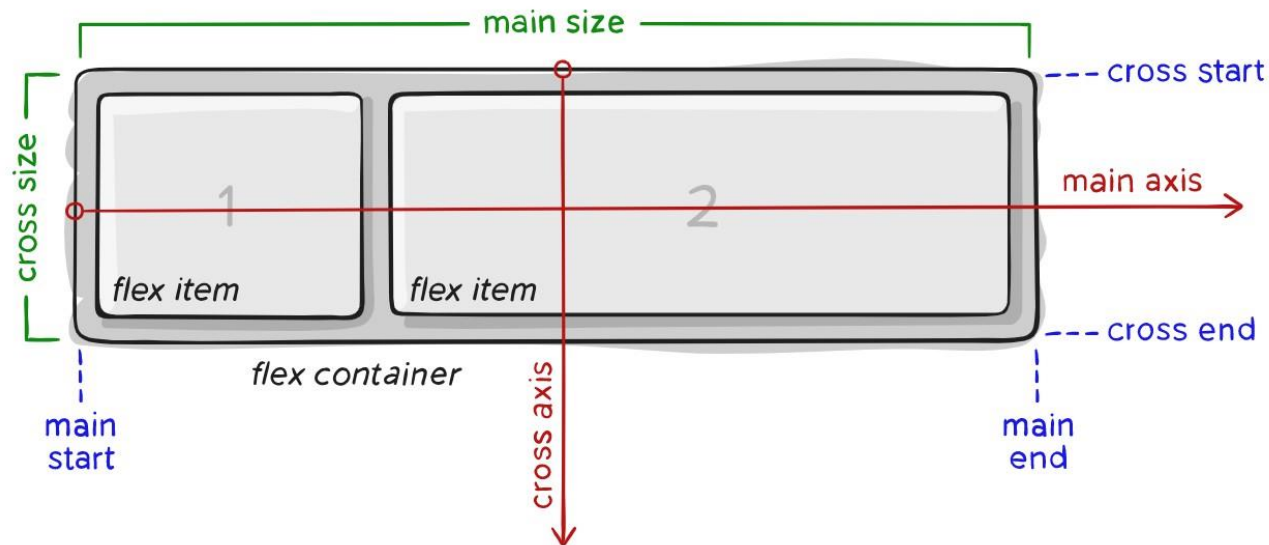
# Flexbox Layout



- The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space.
- Flexbox is for one dimensional layout (row or column).
- set `display` property to `flex` on the containing element

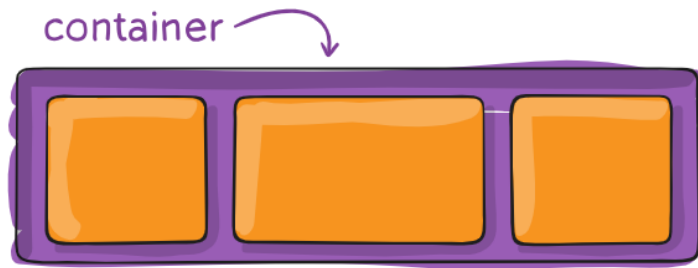
`display: flex;`

- direct child elements are automatically flexible items



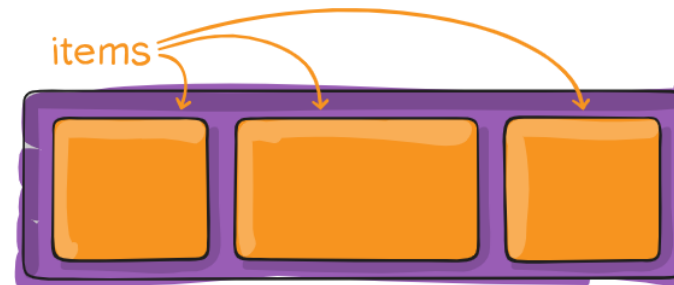
# Flexbox properties

## Properties for the Parent (flex container)



- `display: flex;`
- `flex-direction: row | row-reverse | column | column-reverse;`
- `flex-wrap: nowrap | wrap | wrap-reverse;`
- `flex-flow: <'flex-direction'> | <'flex-wrap'>`
- `justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;`
- `align-items: stretch | flex-start | flex-end | center | baseline;`
- `align-content: flex-start | flex-end | center | space-between | space-around | stretch;`

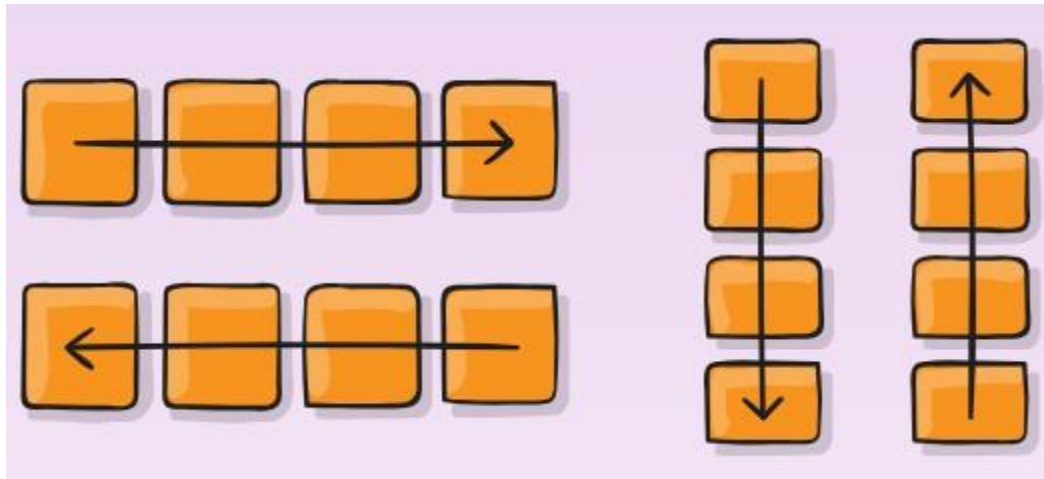
## Properties for the Children (flex items)



- `order: <integer>; /* default is 0 */`
- `flex-grow: <number>; /* default 0 */`
- `flex-shrink: <number>; /* default 1 */`
- `flex-basis: <length> | auto; /* default auto */`
- `flex: none | [ <'flex-grow'> <'flex-shrink'>? | <'flex-basis'> ]`
- `align-self: auto | flex-start | flex-end | center | baseline | stretch;`

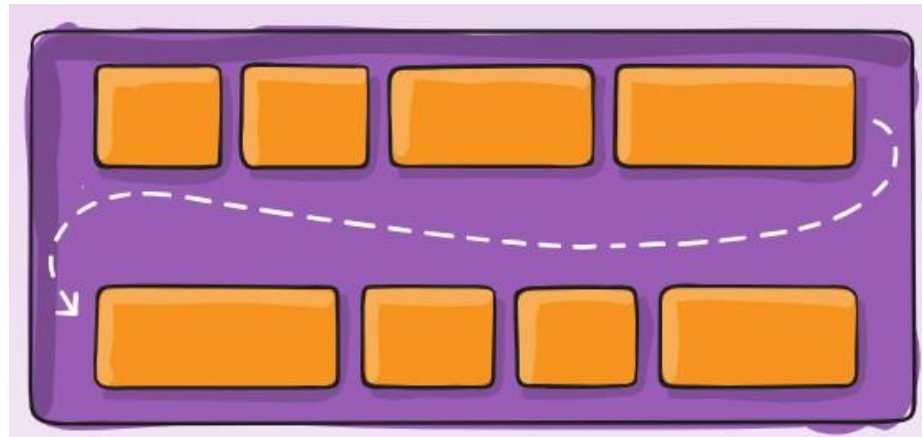
# Flex container: **flex-direction**

- Specifies the direction of the flexible items inside a flex container
  - `row` (default): left to right in ltr; right to left in rtl
  - `row-reverse`: right to left in ltr; left to right in rtl
  - `column`: same as row but top to bottom
  - `column-reverse`: same as row-reverse but bottom to top



# Flex container: **flex-wrap**

- Specifies whether the flex items should wrap or not, if there is not enough room for them on one flex line
- By default, flex items will all try to fit onto one line.
  - `nowrap` (default): all flex items will be on one line
  - `wrap`: flex items will wrap onto multiple lines, from top to bottom.
  - `wrap-reverse`: flex items will wrap onto multiple lines from bottom to top.



# flex container properties

- [flex direction](#): column or row
- [flex wrap](#): wrap or nowrap
- [center vertically and horizontally](#) via justify-content and align-items
- [Responsive image grid](#)
- [Responsive website](#)

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

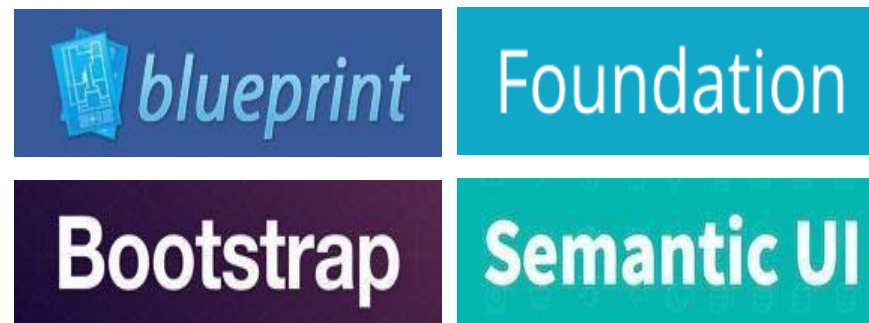
```
.grid-container {  
  grid-template-areas:  
    'header header header header header header'  
    'menu main main main right right'  
    'menu footer footer footer footer footer';  
}
```

good example and exercise [https://www.w3schools.com/css/css\\_templates.asp](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">this column will be 4 of 12 grid elements </div>

<div class="col-sm-8"> this column will be 8 of 12 grid elements </div>

</div>

- col-lg, stack when the width is < 1200px. (columns always pretty large)
- col-md, stack when the width is < 992px.
- col-sm, stack when the width is < 768px.
- col-xs, then the columns will never stack. (columns might get very small)

Bootstrap : <https://getbootstrap.com/>

Bootstrap theme: <https://bootswatch.com/default/>

# 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 flex and grid 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.

