CS472 Web Programming
Lecture 3: Layout

Maharishi University of Management -Fairfield, Iowa

© 2019

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.

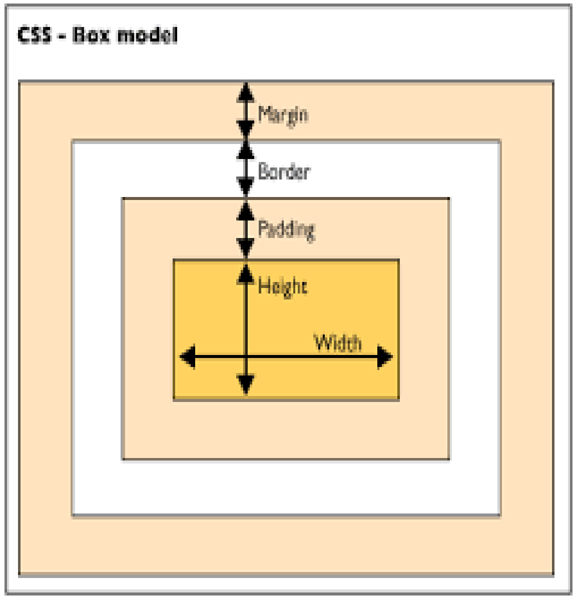
The box model is another encapsulation mechanism that allows layout style to be separate from the page content. Life is found in layers.

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)

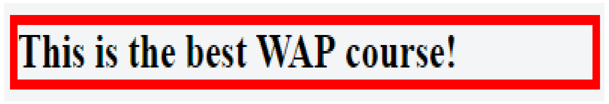
- width = content width + L/R padding + L/R border + L/R margin
- height = content height + T/B padding + T/B border + T/B margin

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



CSS properties for borders

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



This is the best WAP course!

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

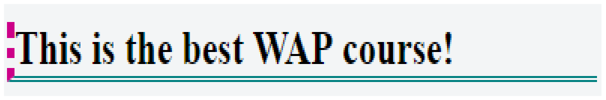
- **thickness** (specified in px, pt, em, or thin, medium, thick)
- **style** (none, hidden, dotted , dashed , double , groove , inset , outset , ridge , solid)
- **color** (specified as seen previously for text and background colors)

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

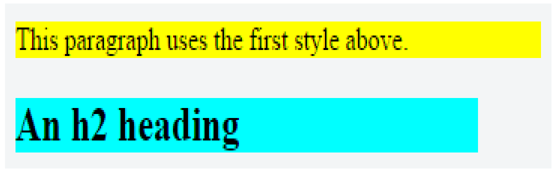
```
p {
 border-left: thick dotted #CC0088;
 border-bottom-color: rgb(0, 128, 128);
 border-bottom-style: double;
}
```


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.
- 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



This paragraph uses the first style above.

An h2 heading

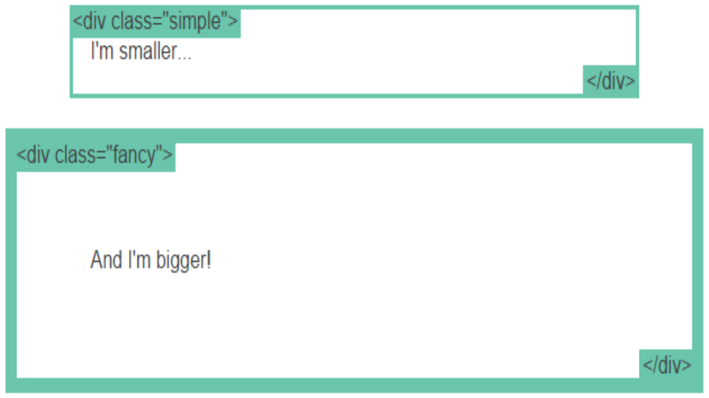
width, height, max-width, max-height, min-width, min-height

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

The Box Model Caveat

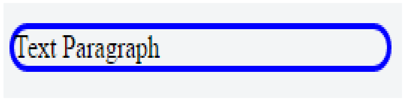
When you set the **`width`** of an element, the element can actually appear bigger than what you set: the element's **`border`** and **`padding`** will stretch out the element beyond the specified width.

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



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:

```
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;
}
```

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;`

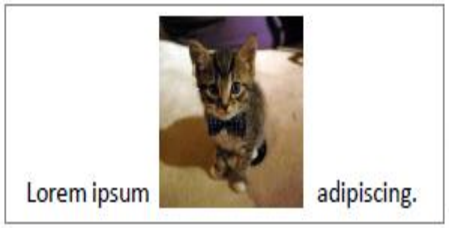
Details about `inline` boxes

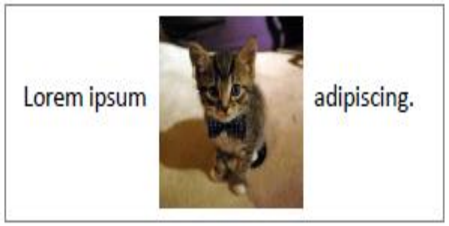
- size properties (width, height, min-width, etc.) are ignored for inline boxes
- margin-top and margin-bottom are ignored, but margin-left and margin-right are not
  - Padding top and bottom ignored
- each inline box's vertical-align property aligns it vertically within its block box
- **text-align** describes how inline content is aligned in its parent block element.
  - does not control the alignment of block elements, only their inline content

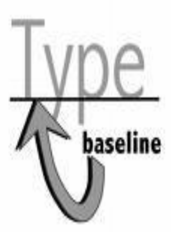
The `vertical-align` property

- 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
- image is vertically aligned to the baseline of the paragraph, which isn't the same as the bottom

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





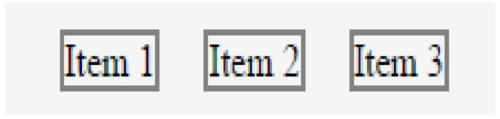


Displaying **block** elements as **inline**

- 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;
}
```



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;`

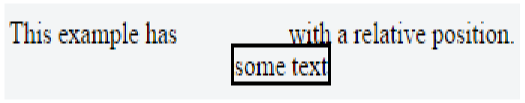
- **static** is the default position value for all elements.
- An element with **position: static;** is not positioned in any special way.
- A static element is said to be not positioned and an element with its position set to anything else is said to be positioned.

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;
}
```


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.

```
#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;
}
```

- 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.

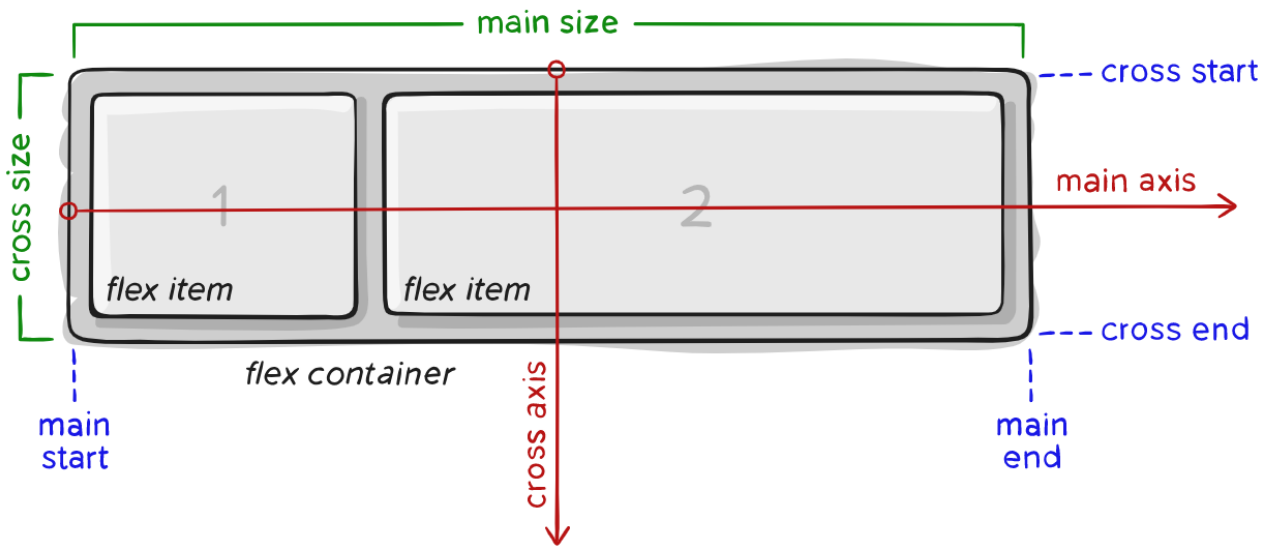
Main Point

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.

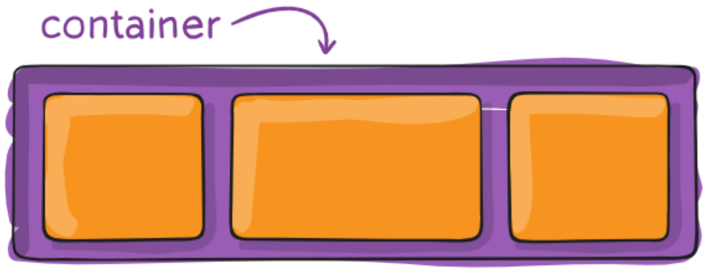
Flexbox Layout

- 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



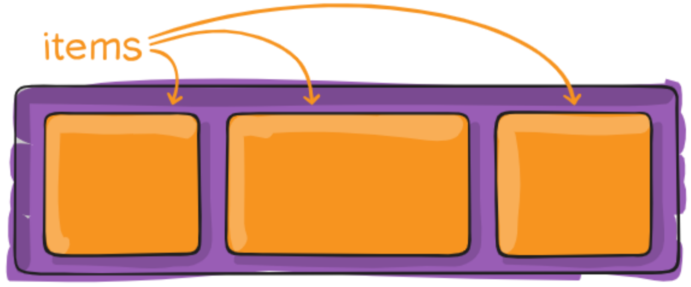
Flexbox properties

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)

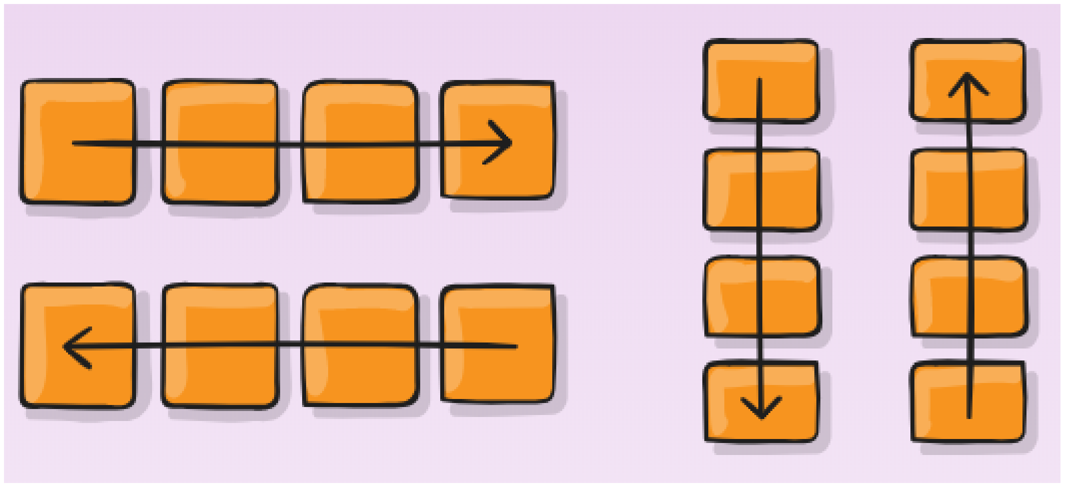


- order: <integer>; /* default is 0 */
- flex-grow: <number>; /* default 0 */
- flex-shrink: <number>; /* default 1 */
- flex-basis: <length> | auto; /* default auto */
- flex: none | [<'flex-grow'> <'flex-shrink'>? || <'flex-basis'>]
- align-self: auto | flex-start | flex-end | center | baseline | stretch;

Credit: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

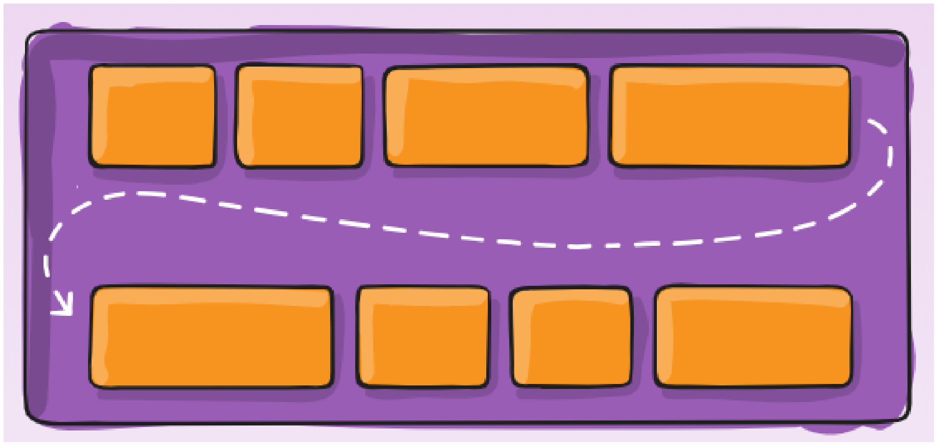
Flex container: **flex-direction**

- 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.



CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE
CSS 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.