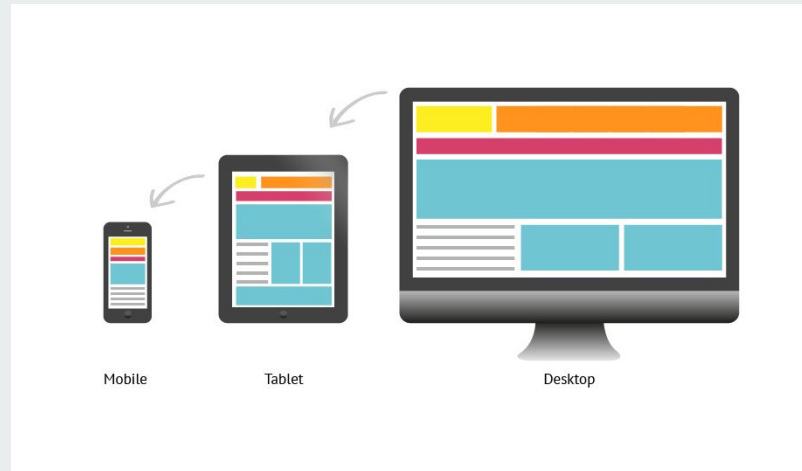


CS 201: Week 3

Responsive Web Design



Responsive Design

- Responsive web design makes your web page look good on all devices.
- Responsive web design uses only HTML and CSS for creating a flexible layout
- Javascript can be used to enhance but design should not be dependent on JS



Design Strategies: Mobile First

Mobile First means **designing for mobile before designing for desktop** or any other device (This will make the page display faster on smaller devices).

Instead of changing styles when the width gets smaller, we should change the design when the width gets larger. This will make our design Mobile First and load faster on mobile devices.



Make your page responsive

- -The **fluid grid** concept calls for page element sizing to be in relative units like percentages, rather than absolute units like pixels or points.
- -**Flexible images** are also sized in relative units, so as to prevent them from displaying outside their containing element. *ie `img {max-width: 100%;}`
- -**Media queries** allow the page to use different CSS style rules based on characteristics of the device the site is being displayed on, most commonly the width of the browser.



Media Queries



Media query is a CSS technique introduced in CSS3. It uses the @media rule to include a block of CSS properties only if a certain condition is true. You can use min or max width for the condition. You may also specify a layout for print or the screen only.

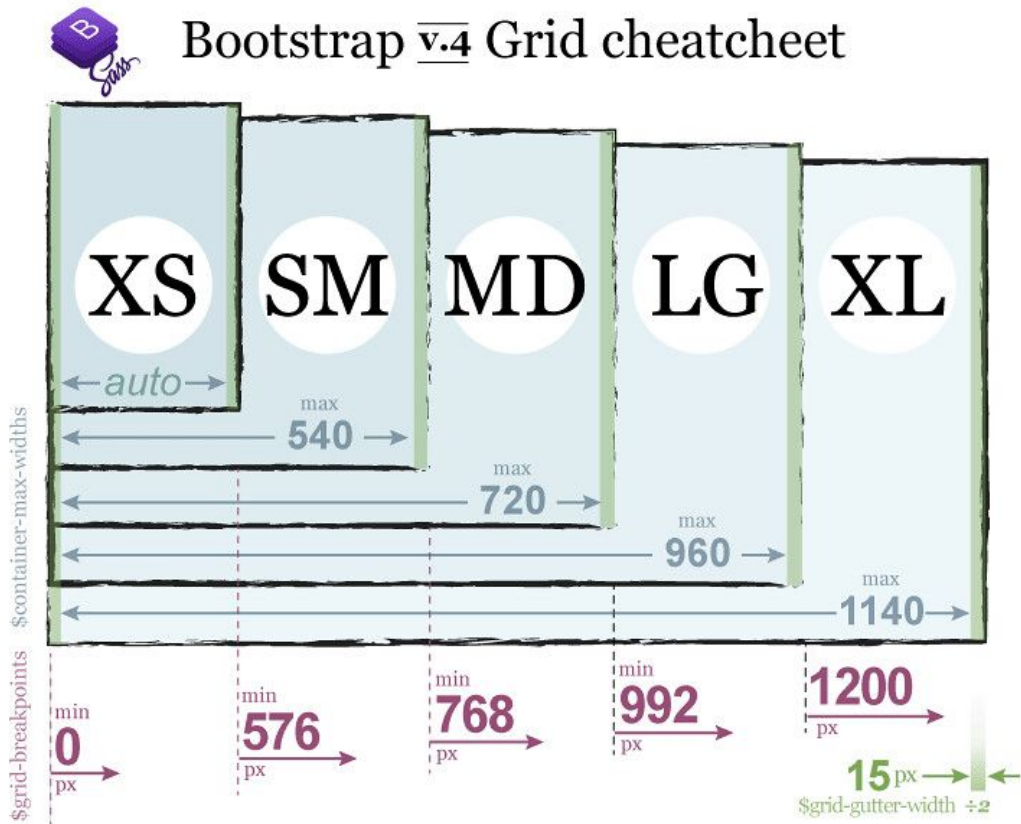
- Media queries are used to implement **breakpoints** in our layout where we can then apply new rules to adjust the layout for a particular viewport:

```
@media only screen and (max-width: 500px) {  
  
    body {  
  
        background-color: lightblue;  
  
    }  
  
}
```

Breakpoints

Breakpoints allow us to adjust our layout rules based on different device screen/viewport widths.

Generally you'll want to have at least three different layouts targeting mobile, tablet and desktop/laptop screens.

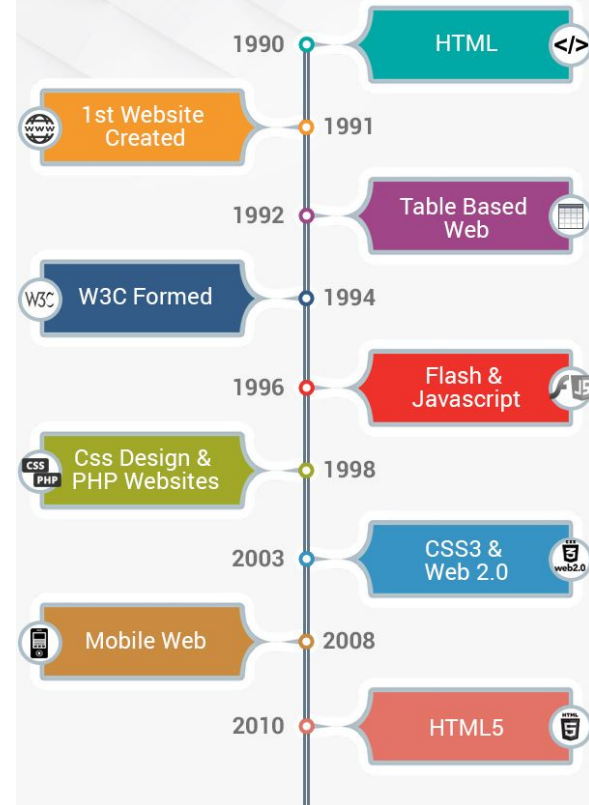


Strategies for Layout

Different techniques have been employed to control the layout of HTML content:

- Tables** (antiquated/avoid)
- Floats** (falling out of favor but used in BS3)
- CSS Flexbox** (modern: used by BS4)
- CSS Grid** (modern: used by BS5 in combo with Flexbox)

The Evolution of WEB DESIGN



Layout: Flexbox vs Grid

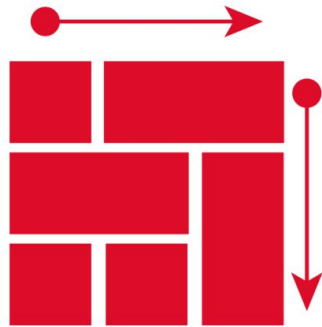
-**CSS Grid** is a CSS layout method developed for two-dimensional layouts. It can manage both columns and rows.

-**Flexbox** is a one-dimensional layout model. It can only handle rows or columns, not both. Flexbox can have multiple rows but doesn't offer the same degree of differing vertical control that Grid has.

-Flexbox focuses on content whereas Grid's main focus is layout. Flexbox, gives a container the ability to alter its items' width, height, and order to fit the available space within the container.



Flexbox
ONE DIMENSION



CSS Grids
TWO DIMENSIONS

CSS Flexbox

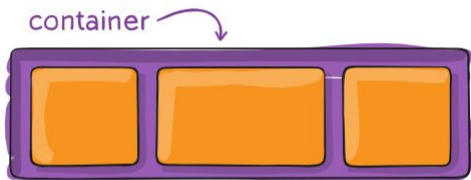
Reference/Graphics:

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

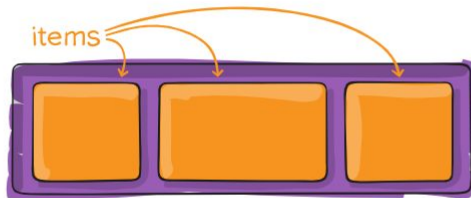
A flex layout gives the container the ability to alter its items' width/height (and order) to best fill the available space (mostly to accommodate to all kind of display devices and screen sizes). A flex container expands items to fill available free space or shrinks them to prevent overflow

The flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based).

Flexbox layout is most appropriate to the components of an application, and small-scale layouts, while the Grid layout is intended for larger scale layouts.



↳ **Properties for the Parent
(flex container)**



↳ **Properties for the Children
(flex items)**

CSS Flexbox: Container Properties

display

This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

flex-direction

This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept.

Think of flex items as primarily laying out either in horizontal rows or vertical columns.

- row (default):
- row-reverse: reverses order of row
- column:
- column-reverse: same as row-reverse but bottom to top



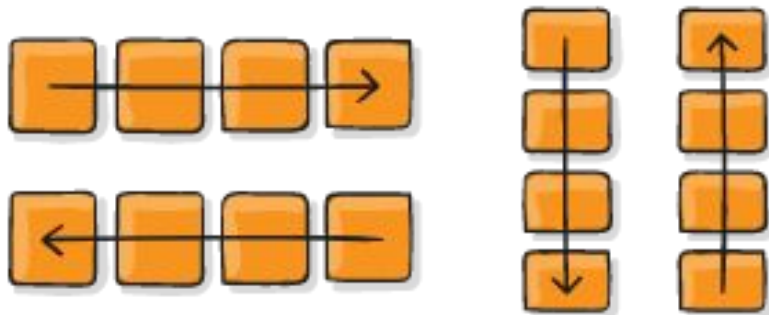
CSS Flexbox: Container Properties

display

This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

flex-direction

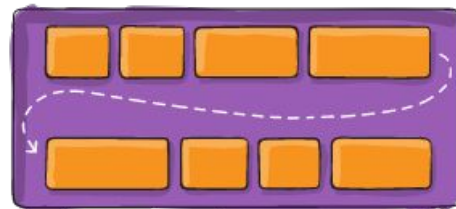
This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.



CSS Flexbox: Container Properties

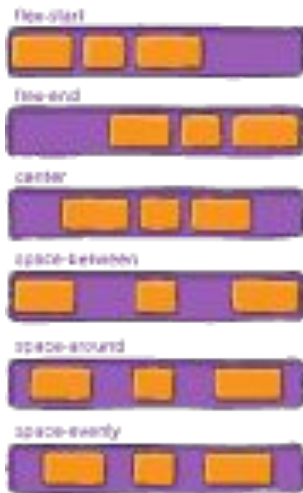
flex-wrap

By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property.



justify-content

This defines the alignment along the main axis. It helps distribute extra free space leftover when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.



CSS Flexbox: Item Properties

flex-basis

This defines the default size of an element before the remaining space is distributed. It can be a length (e.g. 20%, 5rem, etc.) or a keyword. The auto keyword means “look at my width or height property” (which was temporarily done by the main-size keyword until deprecated). The content keyword means “size it based on the item’s content” – this keyword isn’t well supported yet, so it’s hard to test and harder to know what its brethren max-content, min-content, and fit-content do.



flex-basis: 0

0 SETS FLEX ITEMS TAKE UP EXACTLY THE SPACE ALLOTTED BY FLEX-GROW REGARDLESS OF THE CONTENT INSIDE



flex-basis: auto (default)

AUTO SETS FLEX ITEMS TO FIRST CONSIDER CONTENT, AND THEN DISTRIBUTE SPACE AROUND THE CONTENT IN PROPORTION



flex-basis: 0 & flex-basis: 50%

SETTING ONE FLEX ITEM TO A VALUE WILL MAKE THAT ITEM TAKE UP AT LEAST THAT MUCH SPACE, AND THEN REST OF THE SPACE IS DISTRIBUTED ACCORDING TO FLEX-GROW

CSS Flexbox: Item Properties

flex-basis

- Flex-basis controls either “width” or “height” based on the flex-direction
- Flex-basis will override any other *width*: or *height*: properties if specifically declared anything other than flex-basis: **auto** (auto by default)
- The shorthand for flex basis is (flex: \$grow, \$shrink, \$size) and is set to (flex: 0 1 **auto**) by default.
- When flex basis is set to **auto**, it first checks for a width or height property (based on direction) **if none, it uses the elements content sizing**



flex-basis: 0

0 SETS FLEX ITEMS TAKE UP EXACTLY THE SPACE ALLOTTED BY FLEX-GROW REGARDLESS OF THE CONTENT INSIDE



flex-basis: auto (default)

AUTO SETS FLEX ITEMS TO FIRST CONSIDER CONTENT, AND THEN DISTRIBUTE SPACE AROUND THE CONTENT IN PROPORTION



flex-basis: 0 & flex-basis: 50%

SETTING ONE FLEX ITEM TO A VALUE WILL MAKE THAT ITEM TAKE UP AT LEAST THAT MUCH SPACE, AND THEN REST OF THE SPACE IS DISTRIBUTED ACCORDING TO FLEX-GROW

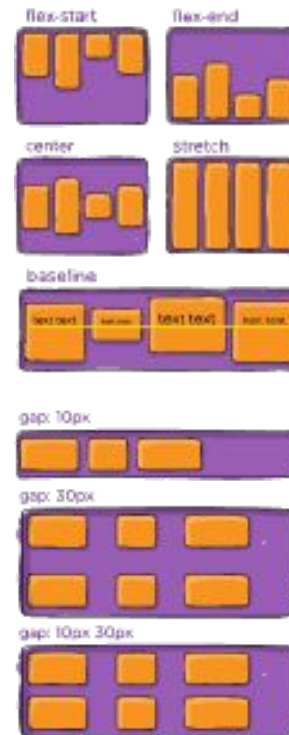
CSS Flexbox: Container Properties

align-content

This aligns a flex container's lines within when there is extra space in the cross-axis, similar to how `justify-content` aligns individual items within the main-axis.

Gap, gap-row, gap-column

The gap property explicitly controls the space between flex items. It applies that spacing only between items not on the outer edges. It applies that spacing *only between items* not on the outer edges.



CSS Syntax: Pseudo Classes



A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it: `button:hover {color:red;}`
- Style visited and unvisited links differently
- Style an element when it gets focus

The syntax of pseudo-classes:

```
selector:pseudo-class {  
  
    property:value;  
  
}
```


CSS Syntax: Pseudo Classes and Elements

:active

a:active Selects the active link

:checked

input:checked Selects every checked `<input>` element

:disabled

input:disabled Selects every disabled `<input>` element

:empty

p:empty Selects every `<p>` element that has no children

:enabled

input:enabled Selects every enabled `<input>` element

:first-child

p:first-child Selects every `<p>` elements that is the first child of its parent

:first-of-type

p:first-of-type Selects every `<p>` element that is the first `<p>` element of its parent

:focus

input:focus Selects the `<input>` element that has focus

:hover

a:hover Selects links on mouse over

:in-range

input:in-range Selects `<input>` elements with a value within a specified range

:invalid

input:invalid Selects all `<input>` elements with an invalid value

:lang(language)

p:lang(it) Selects every `<p>` element with a lang attribute value starting with "it"

:last-child

p:last-child Selects every `<p>` elements that is the last child of its parent

:last-of-type

p:last-of-type Selects every `<p>` element that is the last `<p>` element of its parent

:link

a:link Selects all unvisited links

CSS Syntax: Pseudo Classes and Elements



:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent
:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects <input> elements with no "required" attribute
:required	input:required	Selects <input> elements with a "required" attribute specified
:target	#news:target	Selects the current active #news element
:visited	a:visited	Selects all visited links

CSS Syntax: Pseudo Classes and Elements



A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

Selector	Example	Example description
<u>::after</u>	p::after	Insert something after the content of each <p> element
<u>::before</u>	p::before	Insert something before the content of each <p> element
<u>::first-letter</u>	p::first-letter	Selects the first letter of each <p> element
<u>::first-line</u>	p::first-line	Selects the first line of each <p> element
<u>::marker</u>	::marker	Selects the markers of list items
<u>::selection</u>	p::selection	Selects the portion of an element that is selected by a user

CSS: Colors



Colors in CSS are most often specified by:

- a valid color name - like "red"
- an RGB value - like "rgb(255, 0, 0,[optional alpha])"
- a HEX value - like "#ff0000"

To use an alpha channel you should use RGBA

Use a website like [Colors.co](https://coolors.co) or [Adobe Color](https://color.adobe.com) to create color scheme/palettes



CSS: Position



The **position** property specifies the type of positioning method used for an element.

There are five different position values:

- **Static** (default of HTML elements)
- **relative**
- **fixed**
- **absolute**
- **sticky**

Elements are then positioned using the **top**, **bottom**, **left**, and **right** properties. However, these properties will not work unless the position property is set first. They also work differently depending on the position value.

CSS: Position-Relative



An element with `position: relative;` is positioned relative to its normal position.

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.

We often use this for the parent element which then has an absolutely positioned child

CSS: Position-Fixed



An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

Useful for keeping menus at top of page etc

CSS: Position-Absolute



An element with `position: absolute;` is positioned **relative to the nearest positioned ancestor** (instead of positioned relative to the viewport, like `fixed`).


However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Use `position: relative` on the parent container!

CSS: Position Related Properties



Property	Description
<u>bottom</u>	Sets the bottom margin edge for a positioned box
<u>clip</u>	Clips an absolutely positioned element
<u>cursor</u>	Specifies the type of cursor to be displayed
<u>left</u>	Sets the left margin edge for a positioned box
<u>overflow</u>	Specifies what happens if content overflows an element's box
<u>overflow-x</u>	Specifies what to do with the left/right edges of the content if it overflows the element's content area
<u>overflow-y</u>	Specifies what to do with the top/bottom edges of the content if it overflows the element's content area
<u>position</u>	Specifies the type of positioning for an element
<u>right</u>	Sets the right margin edge for a positioned box
<u>top</u>	Sets the top margin edge for a positioned box
<u>z-index</u>	Sets the stack order of an element



CSS: Overflow



The overflow property has the following values:

- visible - Default. The overflow is not clipped. It renders outside the element's box
- hidden - The overflow is clipped, and the rest of the content will be invisible
- scroll - The overflow is clipped, but a scrollbar is added to see the rest of the content
- auto - If overflow is clipped, a scrollbar should be added to see the rest of the content

It is useful for keep large element/content within a smaller parent container without the overflow resulting in scrollbars or with floats breaking your layout.

CSS: Display Block vs Inline



- The **display** property specifies if/how an element is displayed.
- Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is **block** or **inline**.

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).`<div>`

- `<h1>` - `<h6>`
- `<p>`
- `<form>`
- `<header>`
- `<footer>`

An inline element does not start on a new line and only takes up as much width as necessary.

- ``
- `<a>`
- ``

CSS: Display Inline-Block



- Compared to `display: inline`, the major difference is that `display: inline-block` allows to set a width and height on the element.
- Also, with `display: inline-block`, the top and bottom margins/paddings are respected, but with `display: inline` they are not.
- Compared to `display: block`, the major difference is that `display: inline-block` does not add a line-break after the element, so the element can sit next to other elements.

Newer properties such as flexbox and css grid also allow us to layout elements inline