Lecture 3: CSS Layouts

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 is positioned, 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.

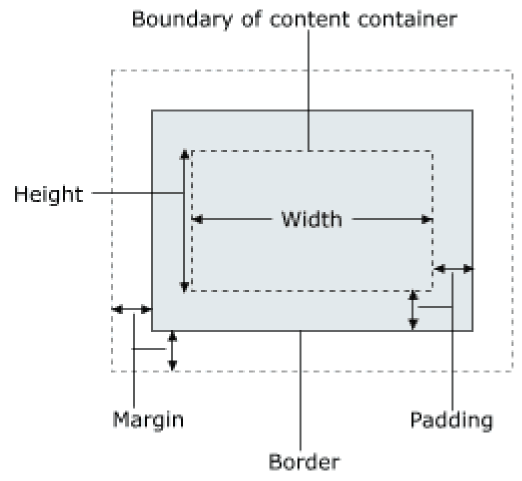
The CSS Box Model

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)

Visual width = content width + L/R padding + L/R border + L/R margin
Visual 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.



Dimensions

For **Block elements 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; }
h2 { width: 50%; }
```

- Using **`max-width`** instead of **`width`** in this situation will improve the browser's handling of small windows. This is important when making a site usable on small devices.
  - If **`width`** is used, when the browser window is smaller than the width of the element, browser than adds a horizontal scrollbar to the page.

Example

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 */
```

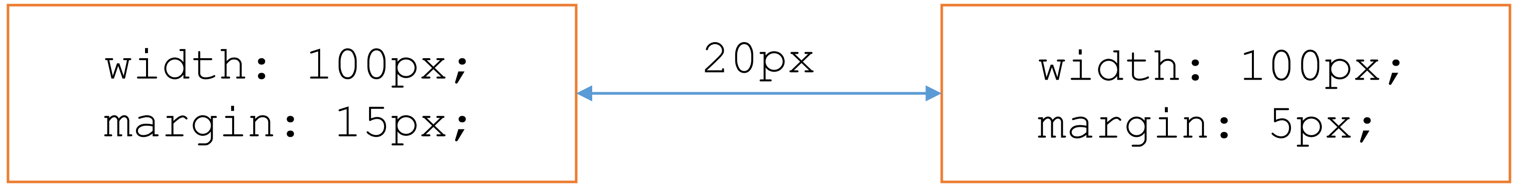
margin-bottom, margin-left, margin-right, margin-top

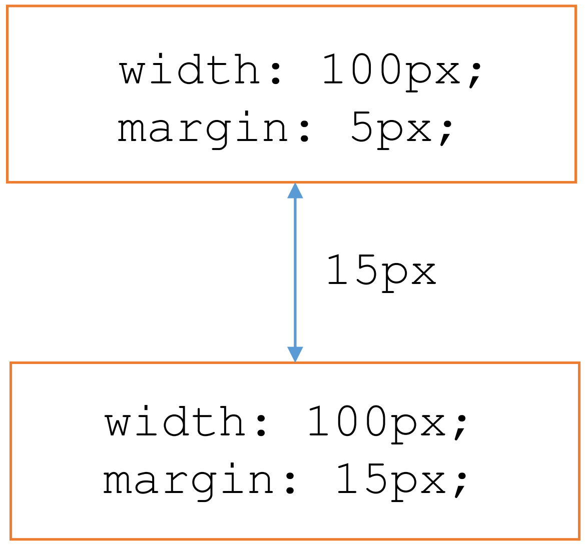
```
h1 { margin: 20px; }
h2 {
    margin-left: 200px;
    margin-top: 30px;
}
```

Margin Collapse

Top and bottom margins 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.

This does not happen on left or right margins! Only on top and bottom margins!



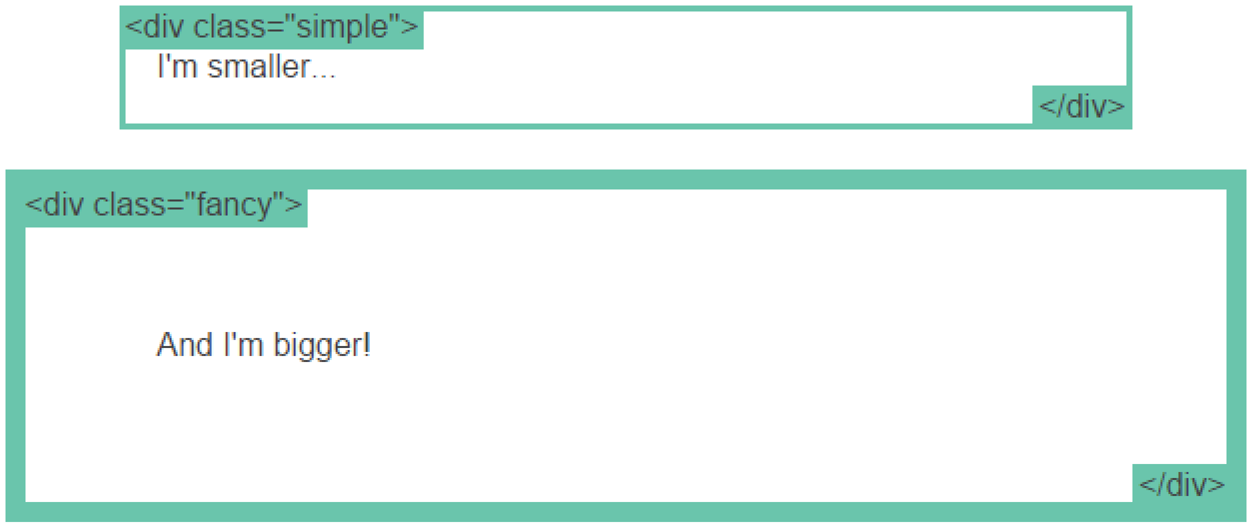


http://www.howtocreate.co.uk/tutorials/css/margincollapsing
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Mastering_margin_collapsing
https://css-tricks.com/what-you-should-know-about-collapsing-margins/

The Box Model Caveat

When you set the **`width`** of an element, the element can 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;
}
```

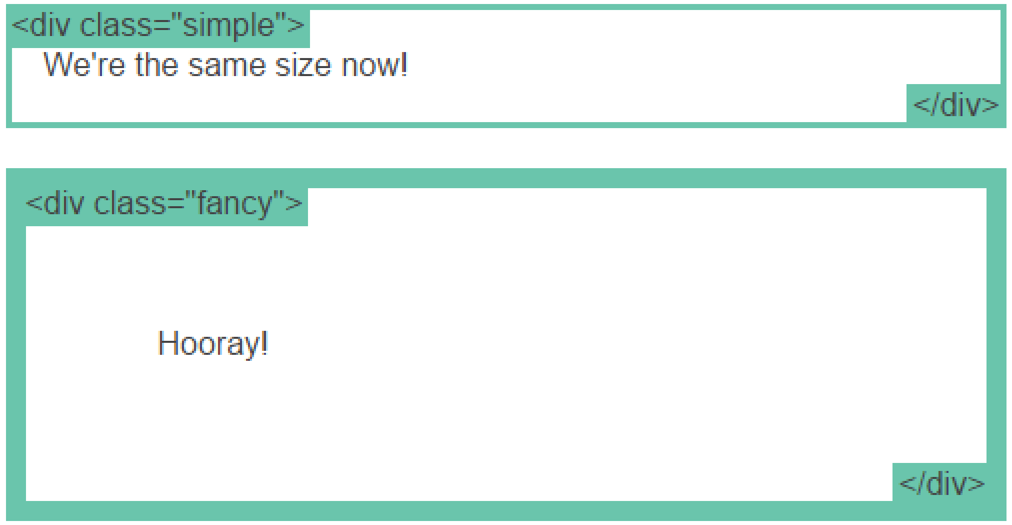


Running example

box-sizing

When you set **`box-sizing: border-box;`** The width and height properties includes the content, the padding and border, but not margin.

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



https://css-tricks.com/box-sizing/#article-header-id-2

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.

Centering a block element vs inline element

- Works only if width is set (otherwise, it will occupy entire width of page)

```
p {
   margin-left: auto;
   margin-right: auto;
   width: 750px;
}
OR
p {
   margin: auto;
   width: 750px;
}
```

- Setting the width of a block-level element will prevent it from stretching out to the edges of its container to the left and right.
- Then, you can set the left and right margins to auto to horizontally center that element within its container.
- The element will take up the width you specify, then the remaining space will be split evenly between the two margins.

- To center text (inline element) within a block element, use

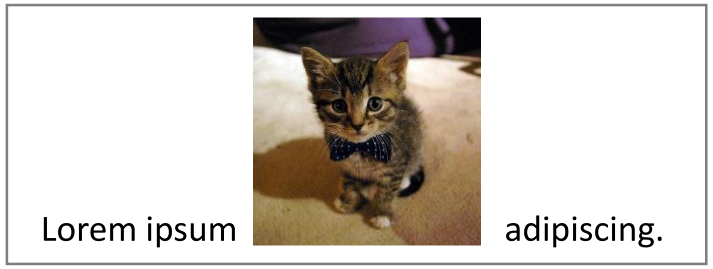
```
p {
   text-align: center;
}
```

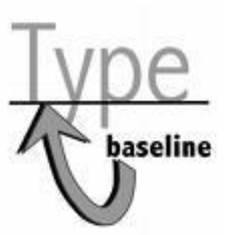
The vertical-align property

Specifies where an **inline element** should be aligned vertically, with respect to other content on the same line.

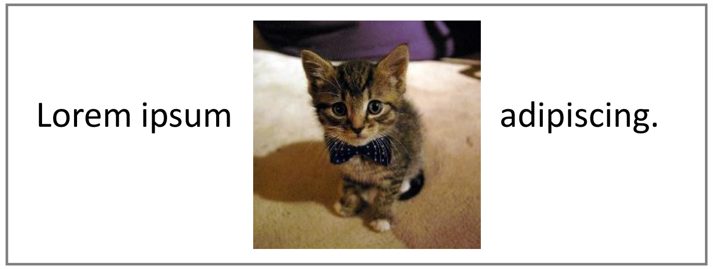
- Can be top, middle, bottom, baseline (default), sub, super, text-top, text-bottom, or a length value or %
  - baseline means aligned with bottom of non-hanging letters

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





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



Block Elements in details

- By default **`block`** elements take the entire width space of the page and the height of its content unless we specify (width, height).
- **`Margin`** and **`Padding`** work as expected
- 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`** element within the page.
    - instead it align text (inline elements) inside the block element.

`inline` Elements in details

- Size properties (**`width`**, **`height`**, **`min-width`**, etc.) are ignored for inline elements. They always get the width and height of their content.
- **`margin-top`** and **`margin-bottom`** are ignored, but **`margin-left`** and **`margin-right`** are not
- The containing block element's **`text-align`** property controls horizontal position of inline elements within it
  - **`left|right|center|justify`**
- Each inline element's **`vertical-align`** property aligns it vertically within its block element
  - **`top|middle|bottom|baseline`**

The `display` property

- 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. However, you can override this.

`Block:` displays an element as a block element (like <p>)

`Inline:` displays an element as an inline element (like <span>)

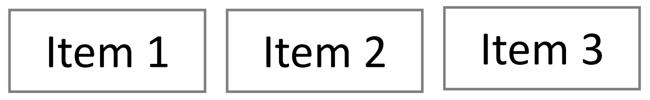
`None`: the element will not be displayed at all

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>
```


Item 1 Item 2 Item 3

```
#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.

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 %

Positioning

`static` is the default value of `position` property. It means that an element isn't positioned. In order to position it, we have to change its predefined type to one of the following:

- **`relative, absolute, fixed, sticky`**

Only then, can we use the offset properties to specify the desired position for our element:

- **`top, right, bottom, left`**

The initial value of these properties is the `auto` keyword.

A positioned element can take advantage of the `z-index` property to specify its stack order.

`position: static;`

`static` is the default position value for all elements.

An element with `position: static;` is not positioned in any special way.

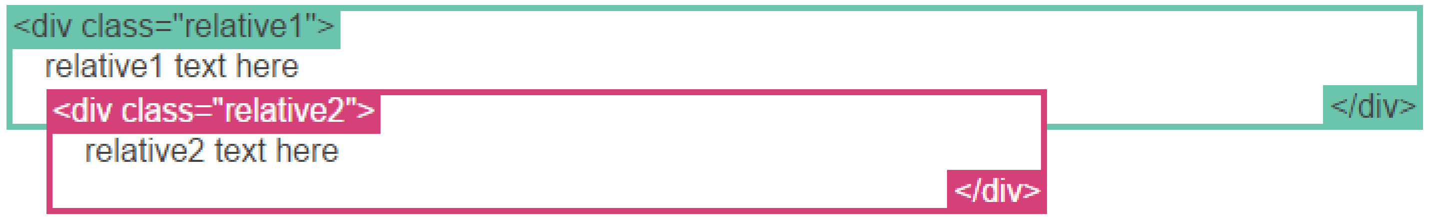
A static element is said to be not positioned and an element with its position set to anything else is said to be positioned.

```
.static { position: static; }
```

position: relative;

relative behaves the same as **static** unless you add some offsets. 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.

```
.relative1 {
   position: relative;
}
.relative2 {
   position: relative;
   top: 20px;
   left: 20px;
   width: 500px;
}
```



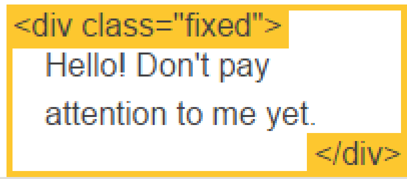
<div class="relative1">
relative1 text here
<div class="relative2">
relative2 text here
</div>
</div>

`position: fixed;`

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

```
.fixed {
   position: fixed;
   bottom: 0;
   right: 0;
   width: 200px;
}
```

A fixed element loses its space in the flow (gets out of normal flow)


```
<div class="fixed">
  Hello! Don't pay
  attention to me yet.
</div>
```

`position: absolute;`

`absolute` behaves like `fixed` except relative to the nearest positioned ancestor (containing element) instead of relative to the viewport and when you scroll the page, it moves.

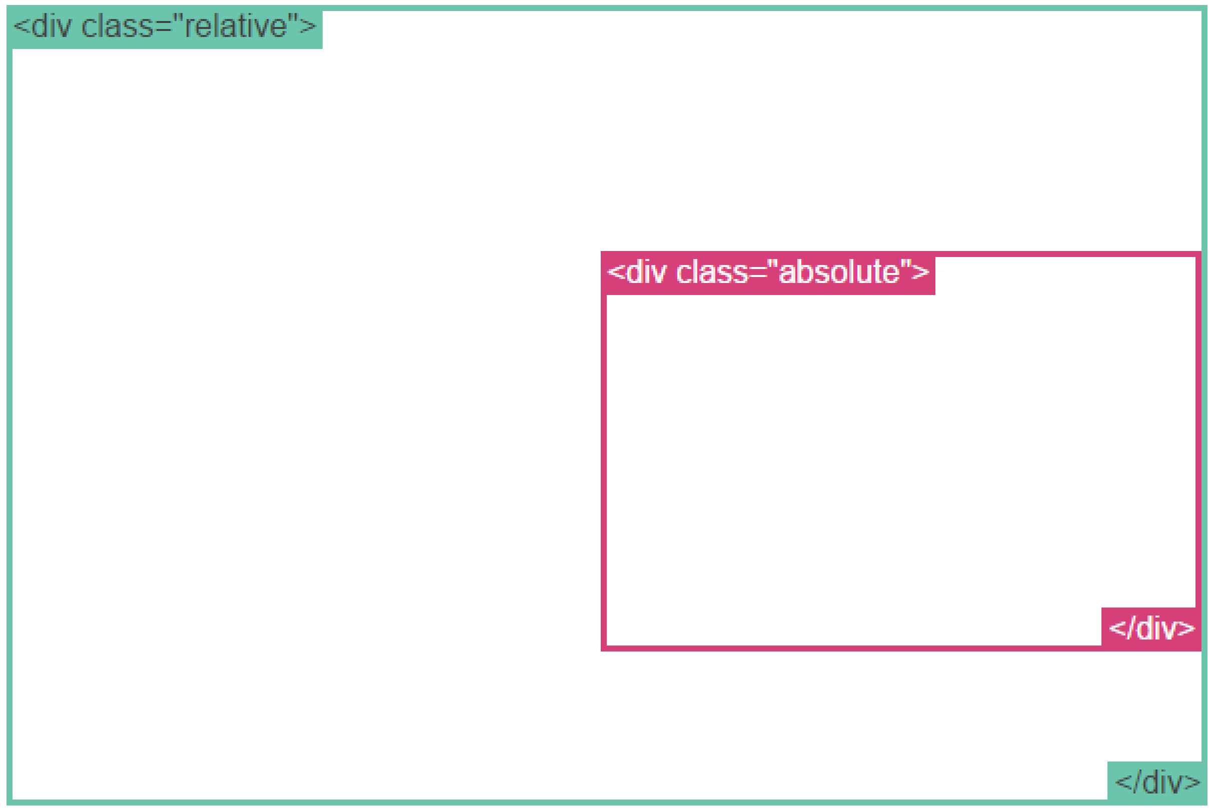
`absolute` element loses its space in the flow (surrounding contents will take its place)

If an absolutely-positioned element has no positioned ancestors, it uses the document body, and still moves along with page scrolling.

Remember, a "positioned" element is one whose position is anything except `static`.

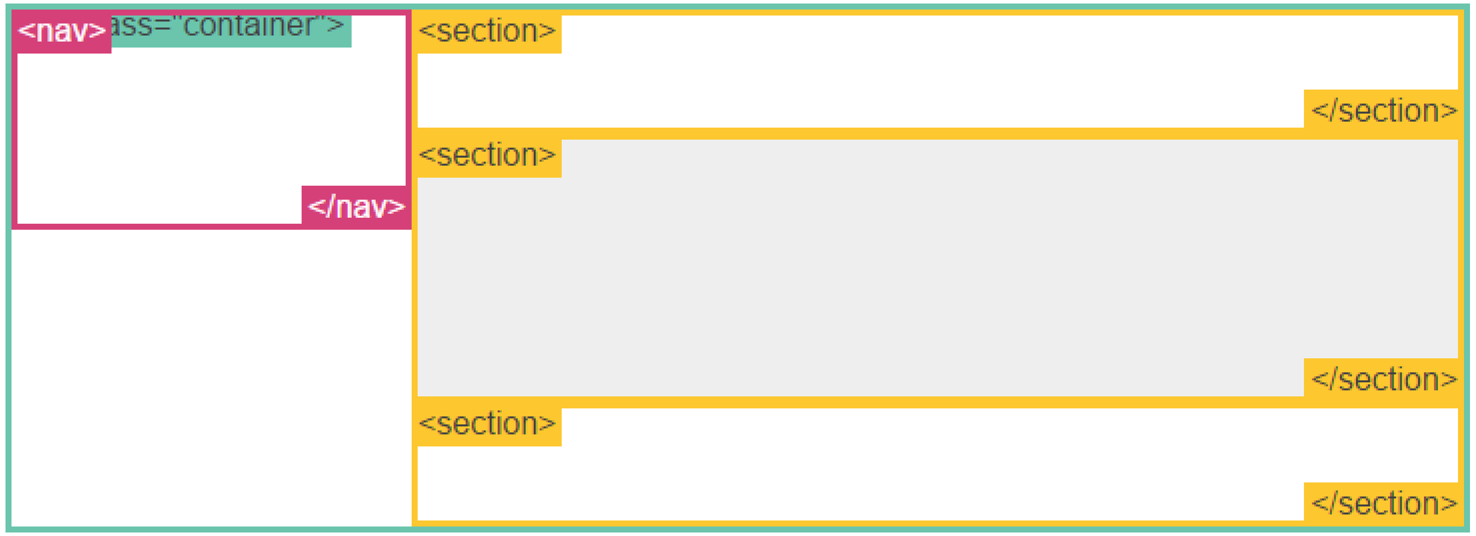
Absolute example

```
.relative {
    position: relative;
    width: 600px;
    height: 400px;
}
.absolute {
    position: absolute;
    top: 120px;
    right: 0;
    width: 300px;
    height: 200px;
}
```



Position Layout Example

```
.container {
   position: relative;
}
nav {
   position: absolute;
   left: 0px;
   width: 200px; }
section {
   margin-left: 200px;
}
```



Positions Review

Relative	Fixed	Absolute
Keep in its original place	Leave the flow (other content will take its place)	Leave the flow
Stay in its original place unless I specify `Top, Right, Bottom, Left`	Move with `Top, Right, Bottom, Left` based on the Viewport	Move with `Top, Right, Bottom, Left` based on the nearest positioned ancestor (not `static`) – if not found: body
`Width` still takes whole screen	`Width` no more takes whole of screen	`Width` no more takes whole of screen

Block: By default takes whole screen `width`, `height` of its content – `width` and `height` can be changed

Inline: By default takes `width` and `height` of its content – `width` and `height` cannot be changed

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.

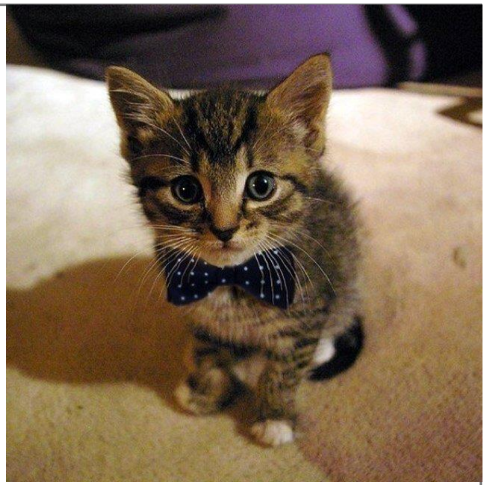
Layouts require understanding how parts fit into a larger whole. The whole is greater than the sum of its parts.

`float`

The float property specifies whether or not an element should float.

In its simplest use, the float element can be used to wrap text around images.

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



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.

- A floating element is removed from normal document flow.
- Underlying text wraps around it as necessary.

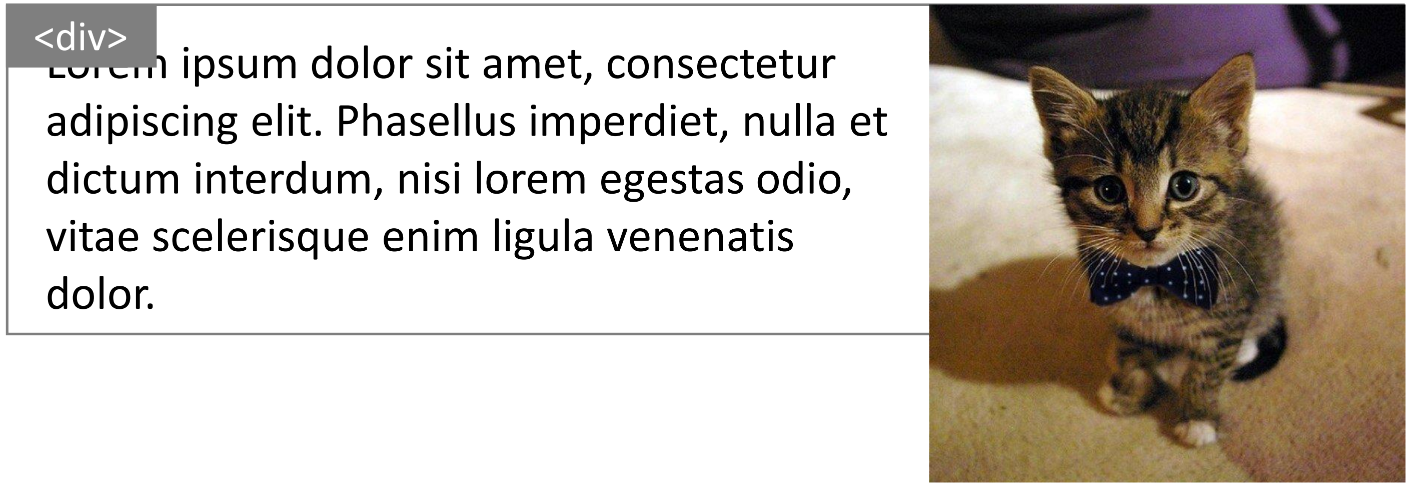
Float and overflow

If an element is taller than the element containing it, and it is floated, it will overflow outside its container.

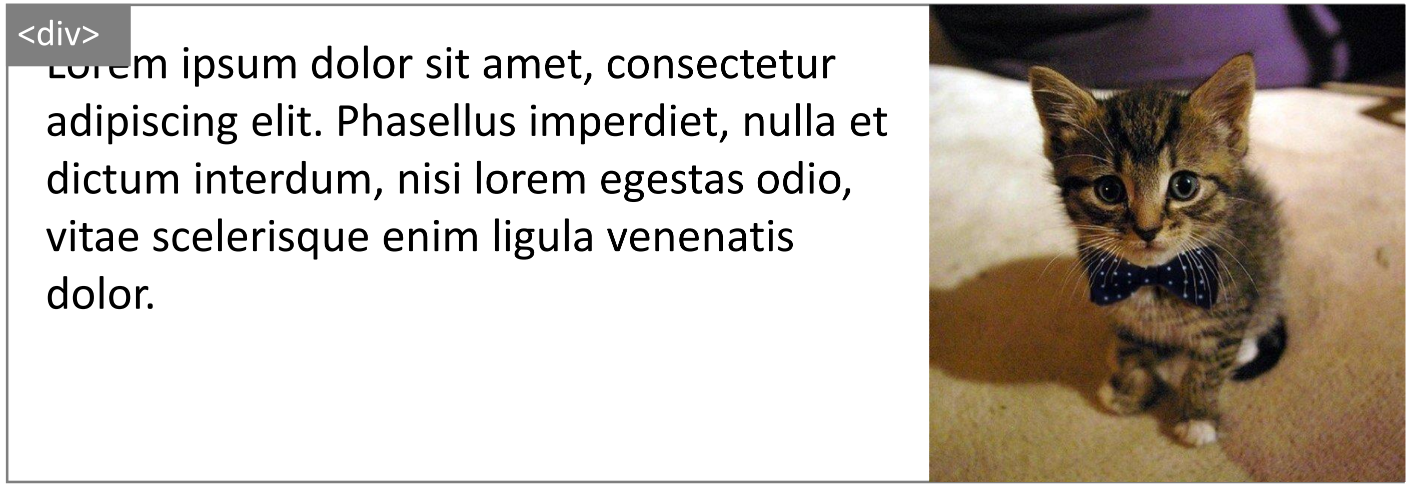
```
img {
    float: right;
}
```

```
img {
    float: right;
}

div {
    overflow: auto;
}
```



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



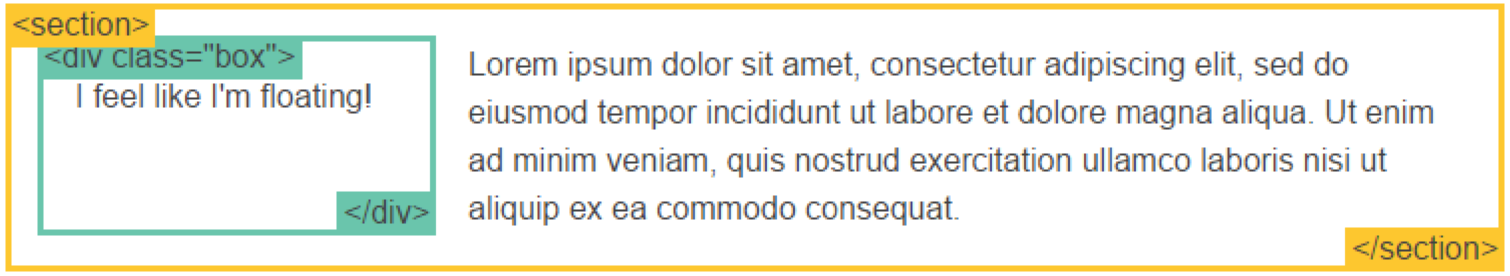
m ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

clear

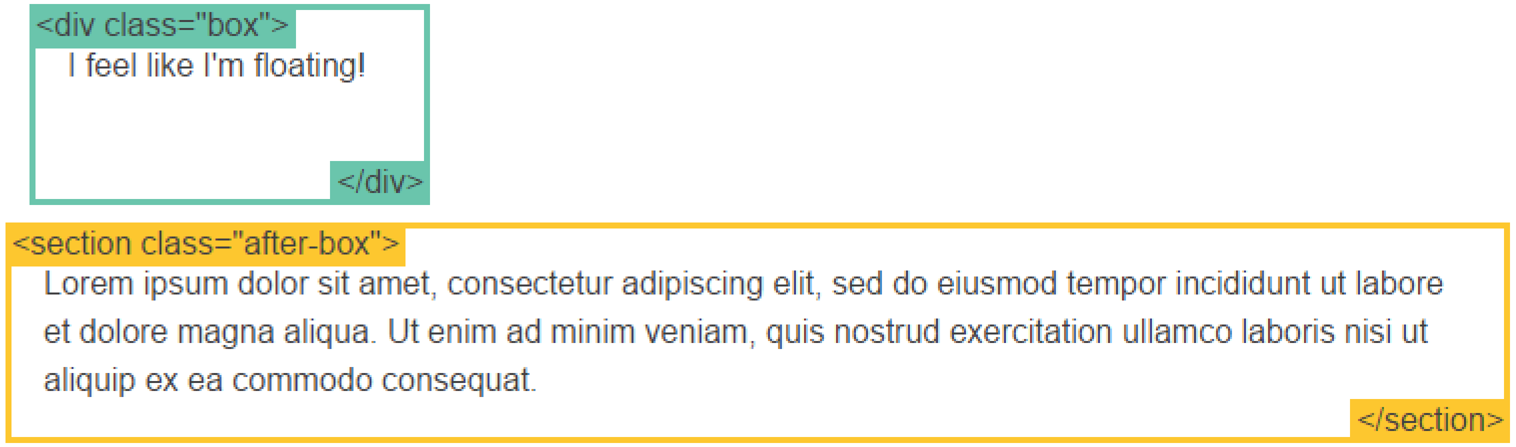
```
<div class="box">...</div>
<section>...</section>

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

The clear property is used to control the behavior of element after a floating element.
Element after a floating element will flow around it. To avoid this, use the clear property.


<section>
<div class="box">
I feel like I'm floating!
</div>
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
</section>

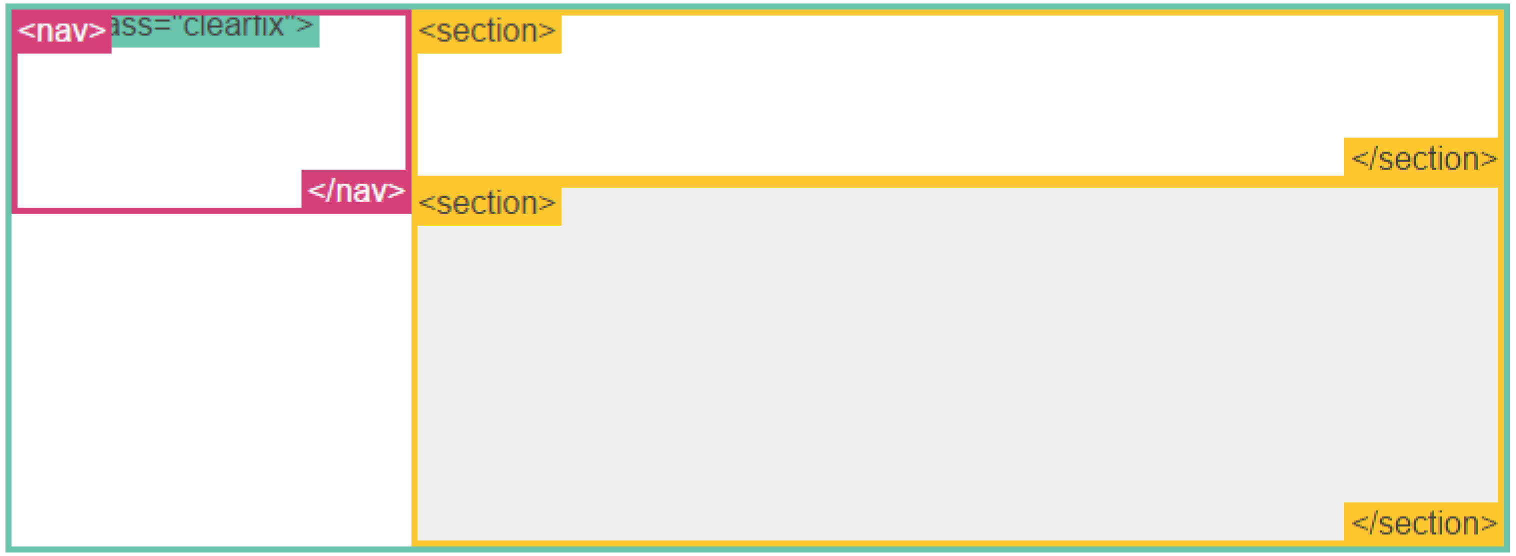
The clear property specifies on which sides of an element floating elements are not allowed to float


<div class="box">
I feel like I'm floating!
</div>
<section class="after-box">
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
</section>

Float Layout Example

```
nav {
    float: left;
    width: 200px;
}
section {
    margin-left: 200px;
}

.clearfix {
    overflow: auto;
}
```

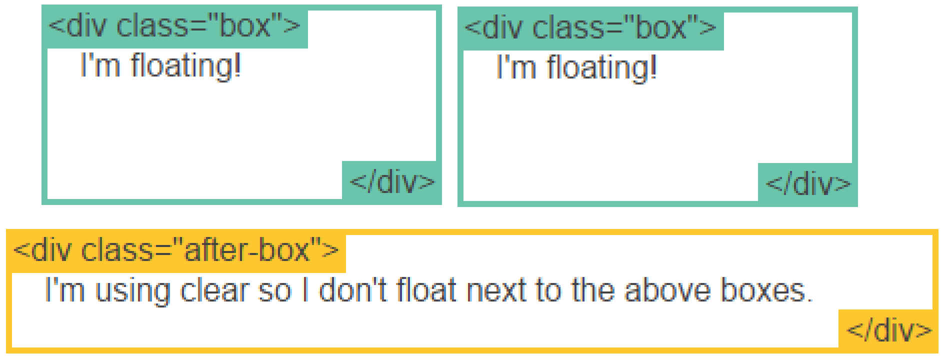


This example works just like the last one.

Notice **`clearfix`** on the container. It's not needed in this example, but it would be if the **`nav`** was longer than the non-floated content.

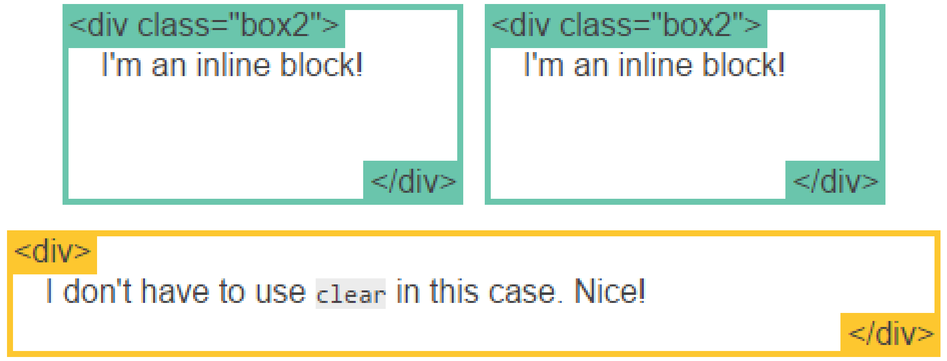
Boxes (Photo Gallery)

The Hard Way (using `float`)



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

The Easy Way (using `inline-block`)

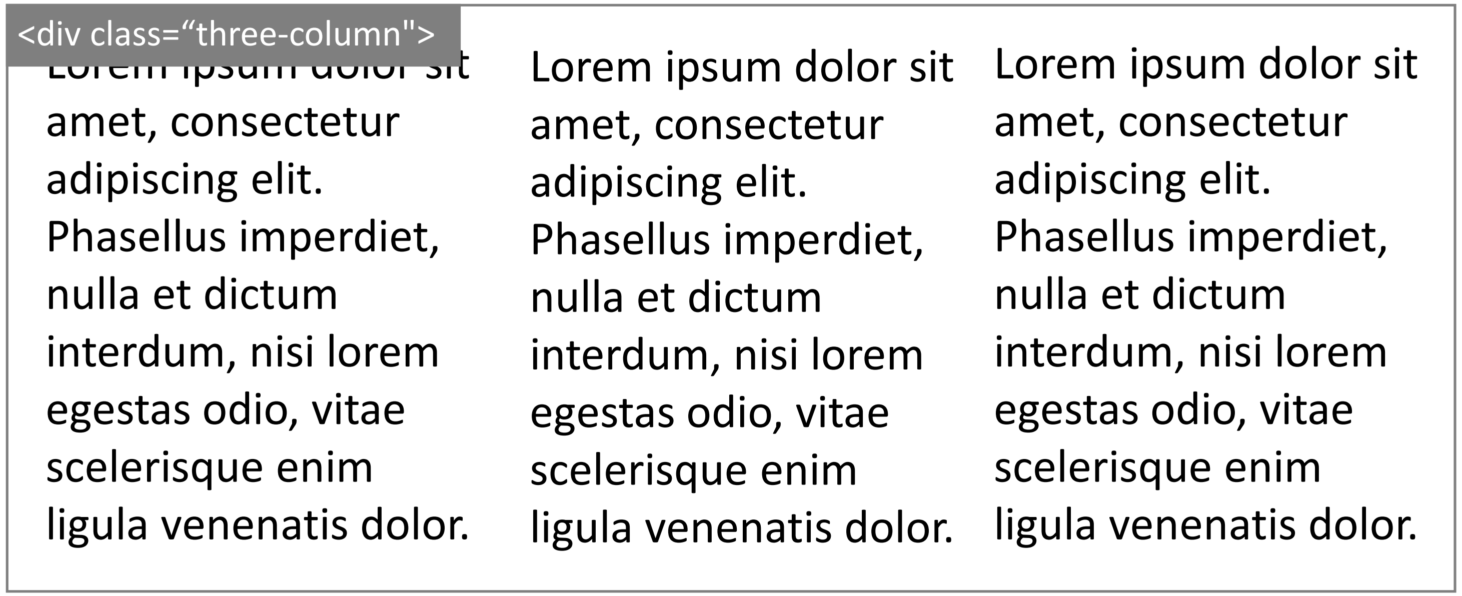


```
.box2 {
    display: inline-block;
    width: 200px;
    height: 100px;
    margin: 1em;
}
```

CSS3 Multiple Columns

This CSS set of properties let you easily make multi-column text.

```
.three-column {
    padding: 1em;
    column-count: 3;
    column-gap: 1em;
}
```



<div class=“three-column">

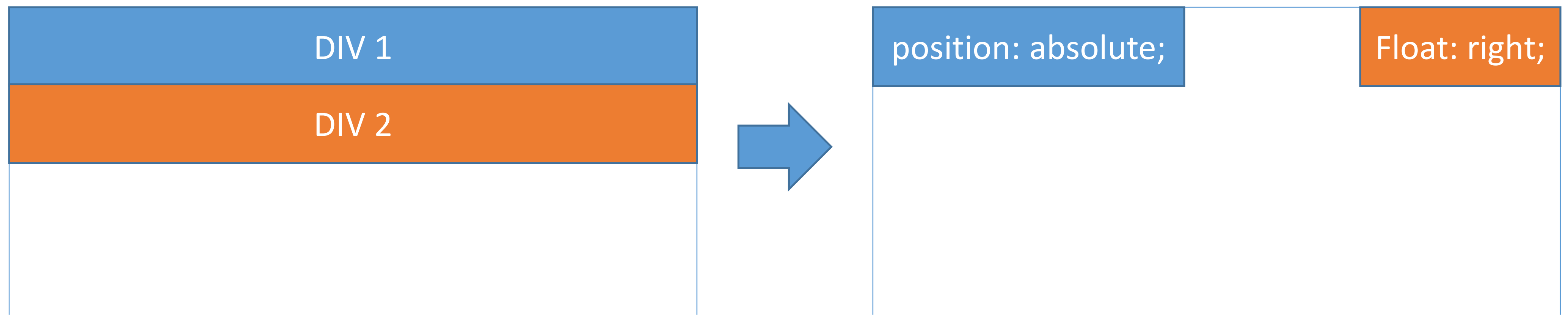
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

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

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

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`**.

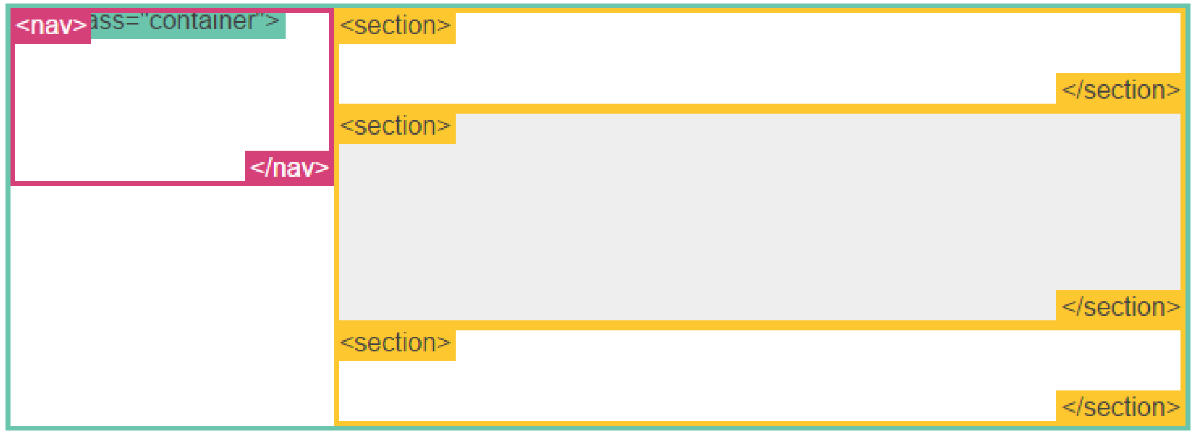


Fluid Layout

You can use percent for layout, but this requires more work.

When this layout is too narrow, the **nav** gets squished.

```
nav {
    float: left;
    width: 25%;
}
section {
    margin-left: 25%;
}
```



Running example

Float Review

1. Leave the flow
2. `Width` is no more takes whole of screen
3. *Sibling* content of a floated element will wrap around it
4. Sequential floated element on the same direction align next to each other but their margin will be determined *by the higher value*
5. Any element that comes after a floated element should be `clear`ed or it will overlap with the floated element (remember point 1)
6. A container of a floated child element will not expand to the height of its child floated element to fix that we give it:
   `overflow: auto.`

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.*

Responsiveness guidelines

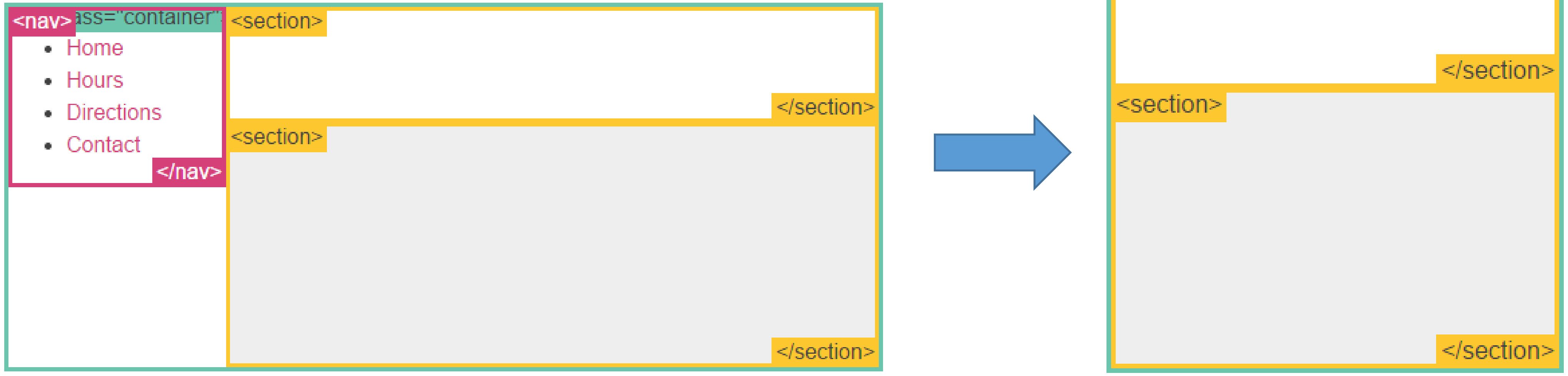
- (Mobile) users scroll websites vertically not horizontally!
- if forced to scroll horizontally or zoom out it results in a poor user experience.
- Some additional rules to follow:
  - Do NOT use large fixed width elements
  - Use CSS media queries to apply different styling for small and large screens
  - Do NOT let the content rely on a particular viewport width to render well

  - https://www.w3schools.com/css/css_rwd_viewport.asp
  - https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag

media queries

Responsive Design is the strategy of making a site that responds to the browser and device width.

```
@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

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

- Narrow screen devices render pages in a virtual window or viewport
  - usually wider than the screen,
  - mobile screen width of 640px, virtual viewport of 980px
  - Users pan and zoom to see different areas of the page
  - Or, browser might shrink the rendered result to fit into the 640px space
  - not good for pages using media queries
  - if the virtual viewport is 980px, queries at 640px or 480px never used
- viewport meta tag lets web developers control the viewport's size and scale.
  - width property controls the size of the viewport.
    - specific number of pixels like width=600
    - device-width, which is the width of the screen
  - View from tablet or phone
    - Without viewport tag
    - With viewport tag

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

- Before the Flexbox Layout module, there were four layout modes:
  - Block, for sections in a webpage
  - Inline, for text
  - Table, for two-dimensional table data
  - Positioned, for explicit position of an element

- The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

- Set display property to flex on the containing element
  - Direct child elements are automatically flexible items.
  - Example

https://www.w3schools.com/csS/css3_flexbox.asp

Flex container properties

Property	Meaning	Values
`flex-direction`	direction the flex items will be stacked on	`row, row-reverse, column, column-reverse`
`flex-wrap`	specifies whether the flex items should wrap or not	`wrap, nowrap, wrap-reverse`
`flex-flow`	shorthand property for setting both `flex-direction` and `flex-wrap` properties	
`justify-content`	align the flex items horizontally	`center, flex-start, flex-end, space-around, space-between`
`align-items`	align the flex item vertically	`center, flex-start, flex-end, stretch, baseline`
`align-content`	align the flex lines (rows of flex items)	`space-between, space-around, stretch, center, flex-start, flex-end`

Perfect Centering: Set both the `justify-content` and `align-items` properties to center, and the flex item will be perfectly centered.

Flex item properties

Property	Meaning	Values
`order`	specifies the order of the flex item	The order value must be a number, default value is 0
`flex-grow`	specifies how much a flex item will grow relative to the rest of the flex items	The value must be a number, default value is 0
`flex-shrink`	specifies how much a flex item will shrink relative to the rest of the flex items	The value must be a number, default value is 0
`flex-basis`	specifies the initial length of a flex item	e.g. 200px
`flex`	shorthand property for `flex-grow`, `flex-shrink`, and `flex-basis`	
`align-self`	specifies the alignment for the selected item inside the flexible container, overrides the default alignment set by the container's `align-items` property	center, flex-start, flex-end

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`
  - An HTML element becomes a grid container when its `display` property is set to `grid` or `inline-grid`
- Direct child element(s) of the grid container automatically becomes grid items and placed inside columns and rows.
  - By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

Example https://www.w3schools.com/css/css_grid.asp

Grid container properties

Property	Meaning	Values
`grid-template-columns`	defines the number of columns in your grid layout, and it can define the width of each column	e.g. `auto auto auto auto;` (4 columns with equal width) 80px 200px auto 40px
`grid-template-rows`	defines the height of each row	e.g. 80px 200px
`grid-column-gap`	sets the gap between columns	e.g. 50px
`grid-row-gap`	sets gap between rows	e.g. 50px
`grid-gap`	shorthand property for the `grid-column-gap` and the `grid-row-gap` properties	e.g. 50px 100px
`justify-content`	horizontally align the whole grid inside the container	`space-evenly, space-around, space-between, center, start, end`
`align-content`	vertically align the whole grid inside the container	`space-evenly, space-around, space-between, center, start, end`

Grid item properties

Property	Meaning	Values

`grid-column`,
`grid-row`
	Defines on which column(s)/ row(s) to place an item.
	You define where the item will start, and where the item will end.
	
	You can refer to line numbers, or use the keyword "span".
		e.g.
		`grid-column: 1 / 5;`
		`grid-row: 1 / 4;`
		
		`grid-column: 1 / span 3;`
		`grid-row: 1 / span 2;`

`grid-area`	shorthand property for the `grid-row-start`, `grid-column-start`, `grid-row-end` and `grid-column-end`	e.g.
		`grid-area: 1 / 2 / 5 / 6;`
		`grid-area: 2 / 1 / span 2 / span 3;`

Naming Grid items

- The `grid-area` property can also be used to assign names to grid items.
- Named grid items can be referred to by the `grid-template-areas` property of the grid container.

```
/* Item1 gets the name "myArea" and spans all five columns in a five columns grid layout: */
.item1 {
    grid-area: myArea;
}

.grid-container {
    grid-template-areas: 'myArea myArea myArea myArea myArea';
}
```

Example

Grid vs Flexbox

- **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.

  - You can turn a grid item into a flex container. You can turn a flex item into a grid.

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.

- Bootstrap
- Blueprint
- Foundation
- SemanticUI

Bootstrap grid system has responsive breakpoints (sm, md, lg, xl)

- Create a row (<div class="row">).
    - add columns (tags with appropriate .col-*-* classes)
    - first star (*) represents the responsiveness: sm, md, lg or xl
        - 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.
    - second star represents a number, which should add up to 12 for each row
    - Example
- Bootstrap 4 has 5 breakpoints, but concept is the same.

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

CSS Positioning

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

---

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