CSS Layout

tl;dr

- Pages have different sections which require some planning to lay them out
- Our options:
  - Tables and absolute positioning are inflexible
  - Floating
  - inline-block
  - flexbox
  - grid

How do you layout pages?

1. Define sections
2. Define sizes
3. Get them to live side-by-side

And how do you get things to live side-by-side?

- Tables
- Absolute positioning
- Inline sections

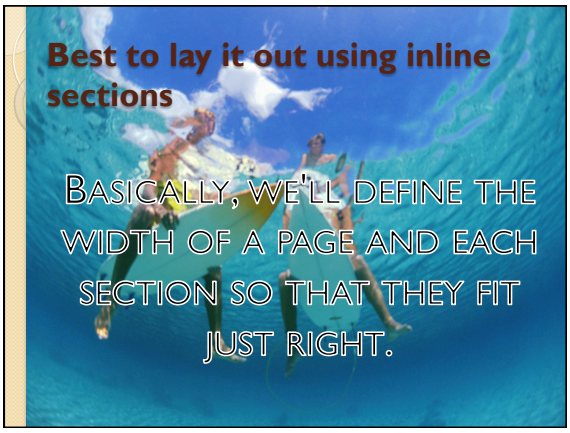
You can lay it out using tables

- Header section: Row 1 with colspan=3
- Links: Row 2, column 1
- Content: Row 2, column 2
- Ads: Row 2, column 3
- Footer: Row 3 with colspan=3
- Easy to understand, but semantically wrong!

You can lay it out using absolute positioning

```
<style>
div {
  position: absolute;
}
div.header {
  height: 100px;
}
div.left {
  top: 100px;
  width: 100px;
}
div.content {
  top: 100px;
  left: 100px;
}
</style>
```

Unfortunately, this does not scale vertically, so if you run out of space, things *occult* one another.



A word about page flow

- Remember, we have two types of elements
- inline elements
  - Text and other things flow around the element
  - No concept of width or height
- block elements
  - A break appears before and after it
  - Has width, height, borders, padding, and margins
- Sounds like we'll need a hybrid of the two

Four ways to hybrid *inline* and *block*

1. floated divs
2. display: inline-block
3. flex boxes
4. grids

Option 1

FLOATED DIVS

Float can do the job

- From the CSS spec:
  - A float is a box that is shifted to the left or right on the current line. The most interesting characteristic of a float (or "floated" or "floating" box) is that content may flow along its side
- Options
  - left/right: no break before if it can fit on the page
  - clear: re-establish breaks

Floating takes some getting used to

- It is the weirdest thing in layouts
- A floated element allows things below it to float up next to it.
- Floating things takes them out of the normal flow of the text.
- When we float s, we allow them to exist side-by-side
- Floated elements are processed in the order that they're presented in the markup.

Hands-on floats

Oh noes! When the browser is narrow, our sections stack on top of one another

- Solution: Fix the width and center it

```
#superDiv {
  width: 960px;
  margin: 0px auto;
}
```

Floats work by aligning their edges

- float: left moves it as far left as it can
- float: right moves it as far right as it can
- They keep stacking until they run out of room
- Then they shift down until they fit

Option 2

DISPLAY: INLINE-BLOCK

The good ...

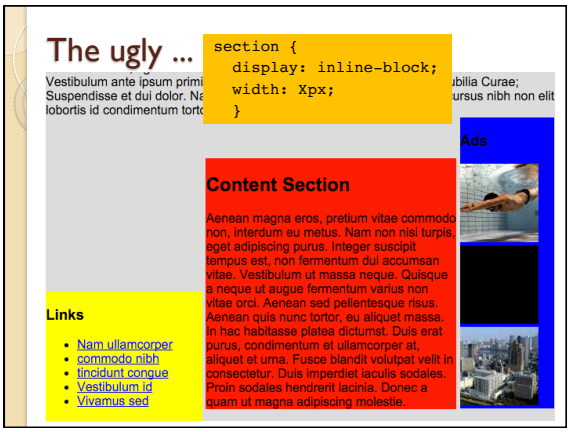
- display: inline-block has the best of both worlds
- Honors width like a block
- Allows side-by-side like an inline

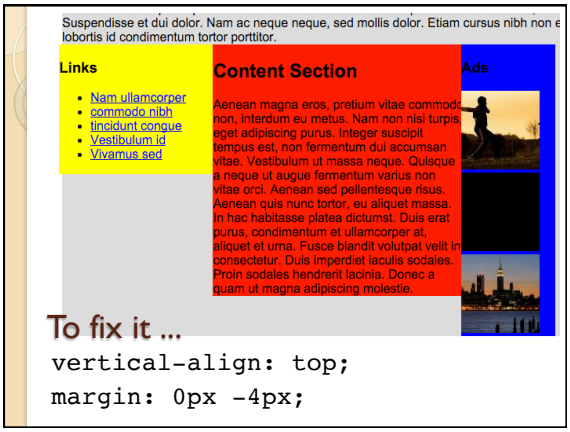
The bad ...

- Alignment and spacing are issues
- They're vertically aligned at the bottom :-(
- There's a space between sections even with no margin. :-(

The ugly …

```
section {
  display: inline-block;
  width: Xpx;
  }
```





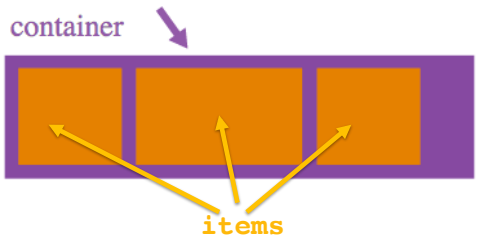
To fix it …

```
vertical-align: top;
margin: 0px -4px;
```

Option 3

FLEX BOXES

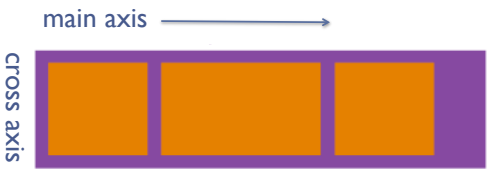
flex involves a container and contents



Mark the container as a flex-box

```
#container {
  display: flex;
}
```

- And best case scenario ... that's all you need!



The container has axes

Most of us would call these
- horizontal
- vertical

The container lays out the items

```
#container {
 flex-direction: column;
 flex-direction: column-reverse;
 flex-direction: row;
 flex-direction: row-reverse;
}
```

flex items have sizes

They have a "favorite" size

flex-basis

They know how to shrink from that and how to grow from that

flex-shrink
flex-grow

Sizes of the items

- flex-basis = the item's "favorite" size.
  - in px, em, %, etc.
- If the viewport is increased from flex-basis ...
- flex-grow = by how much it grows
  - unitless - a relative number
- if the viewport is decreased from flex-basis ...
- flex-shrink= by how much it shrinks
  - unitless - a relative number

To have relative widths/heights set flex-grow/flex-shrink on each item

```
#item1 {
  flex-grow: 3;
}
#item2 {
  flex-grow: 1;
}
#item3 {
  flex-grow: 2;
}
```

- These numbers are unitless and relative to one another.

order

- Default: The order they're in on the page
- Numerically otherwise

Option 4

GRIDS

First, some CSS grid concepts ...

- *containers* have *items*
- containers have *lines* that define rows and columns (aka *tracks*)
- *cells* are where the tracks intersect
- cells can be combined into rectangular *areas*

Just like in flexbox, use a **container**

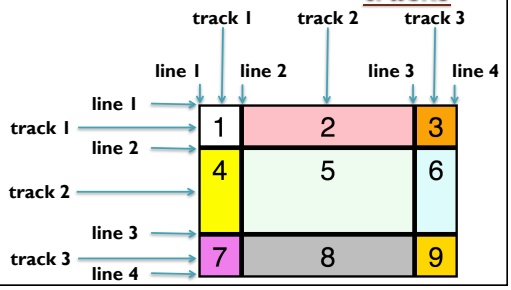
```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
}
```

- A container can be anything in your document:
  - <body>
  -
  - <section>
  - <whatever> (Disclaimer: not a real thing).

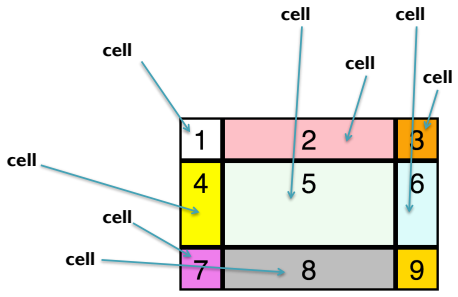
Everything in a grid container is by definition a grid **item**

```
.grid {
  display: grid;
  grid-template-columns: 50px 100px 50px;
  grid-template-rows: 50px 50px 50px;
}
```

grid **lines** separate the grid **tracks**

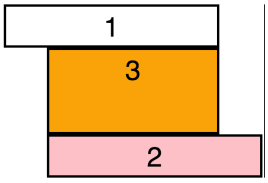


And of course grid **cells** are where the tracks (rows and columns) intersect



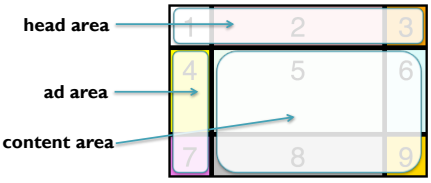
```
.grid {
  display: grid;
  grid-template-columns: 50px 100px 50px;
  grid-template-rows: 50px 50px 50px;
}
#one {
  grid-row: 1/3;
  grid-column: 1/2;
}
#two {
  grid-row: 3/4;
  grid-column: 2/4;
}
#three {
  grid-row:2/3;
  grid-column: 2/3;
}
```

To assign to cells, place between **lines**



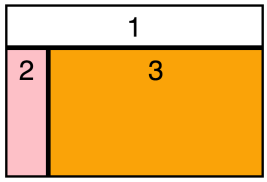

```
.grid {
  display: grid;
  grid-template-columns: 50px 100px 50px;
  grid-template-rows: 50px 50px 50px;
  grid-template-areas:
    'head head    head'
    'ad   content content'
    'ad   content content'
}
```

grid **areas** are optional groups of cells




```
.grid {
  display: grid;
  grid-template-columns: 50px 100px 50px;
  grid-template-rows: 50px 50px 50px;
  grid-template-areas:
    'head head    head'
    'ad   content content'
    'ad   content content'
}
#one {
  grid-area: head;
}
#two {
  grid-area: ad;
}
#three {
  grid-area: content;
}
```

To assign to areas ...



So … which one do I learn?

FLOAT VS INLINE-BLOCK VS GRID VS FLEXBOX

You need all of them!

- Just about any layout can be done with any choice.
- Float is when you need 2 elements to be side-by-side. Not best for full pages anymore.
- Inline-block is best when you want 3 or more page sections to be side-by-side
- Use flexbox for 1-dimensional possibly responsive layouts (ie. rows or columns)
- Use grids for 2d layouts (ie. rows and columns)
- Save grids for complex layouts.

tl;dr

- Pages have different sections which require some planning to lay them out
- Our options:
  - Tables and absolute positioning are inflexible
  - Floating
  - inline-block
  - flexbox
  - grid