Interesting things I learnt about layout

(and general CSS...)

Chen Hui Jing / @hj_chen




Surname First name

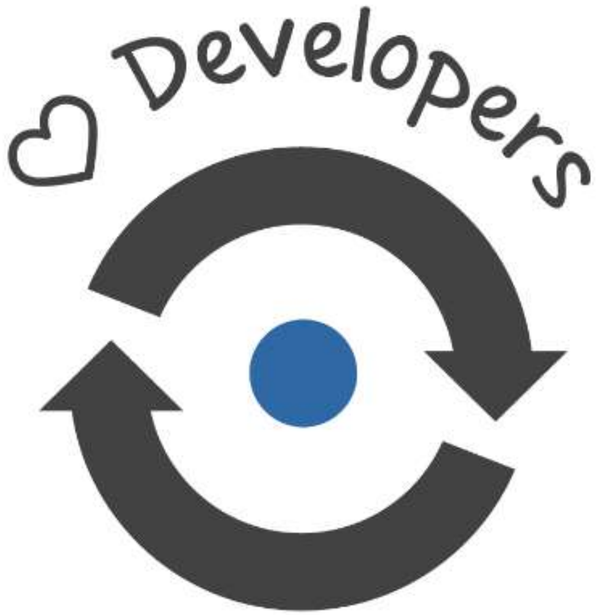
陈	慧 晶
Chen	Hui Jing






@hj_chen



Developer Advocate

nexmo®

The Vonage® API Platform

http://bit.ly/convo-melb

🇸🇬 SingaporeCSS 🇸🇬

https://singaporecss.github.io

@SingaporeCSS | @hj_chen | @wgao19

Initial value of `display` for all elements is `inline`

Then how come `<div>`s, paragraphs, lists and the like are `<display: block>`?

1.00

Because browser default stylesheets.

- Firefox stylesheet: `resource://gre-resources/html.css`
- Link to Chromium stylesheet

Inline-level element behaviour

- `inline`, `inline-table`, `inline-block`, `inline-flex`, `inline-grid`
- `width` and `height` property does not apply
- height of content is based on font size
- `vertical-align` property only applies to inline-level and table-cell elements
- Only margins, borders and paddings along the inline axis have any visible effect on an inline box

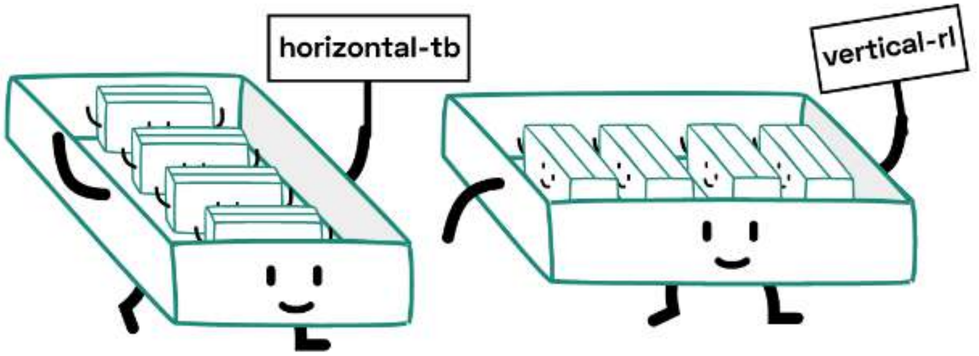

```
<p class="line-container">If an
element <em>generates zero boxes</em>,
was it <strong>really there</strong>
at all?</p>
```

```
}

.linebox .line-container em {
  background-color: forestgreen;
  padding: 0.5em;
}
```

Block formatting contexts

The context that **block-level** boxes participate in

Boxes are laid out one after another, in the block flow direction, from the start of the containing block



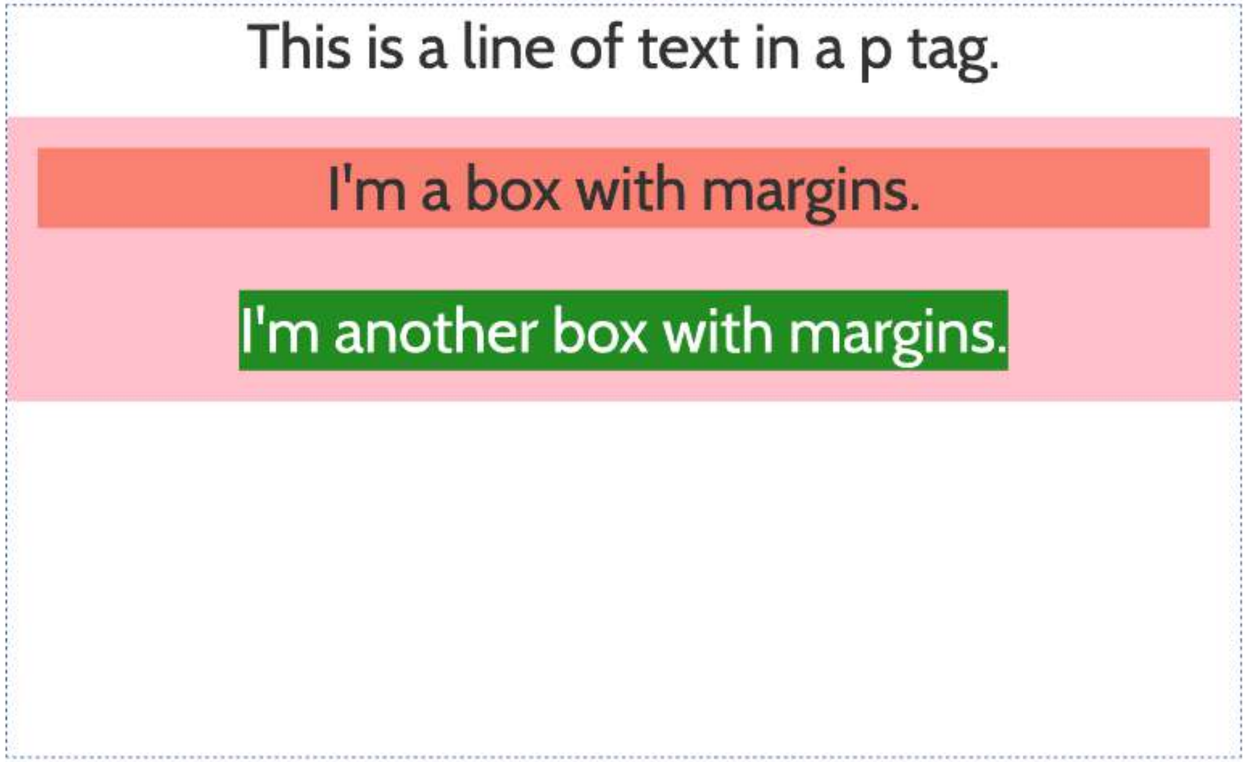
Margins along the *block flow direction* between **adjacent block-level** boxes in the **same** block formatting context collapse

What establishes new block formatting contexts?

- Floats
- Absolutely positioned elements
- Block containers that are **not** block boxes
- Block boxes with `overflow` **other than** `visible`
- Boxes with `display` set to `flow-root`

We need a new BFC because...?

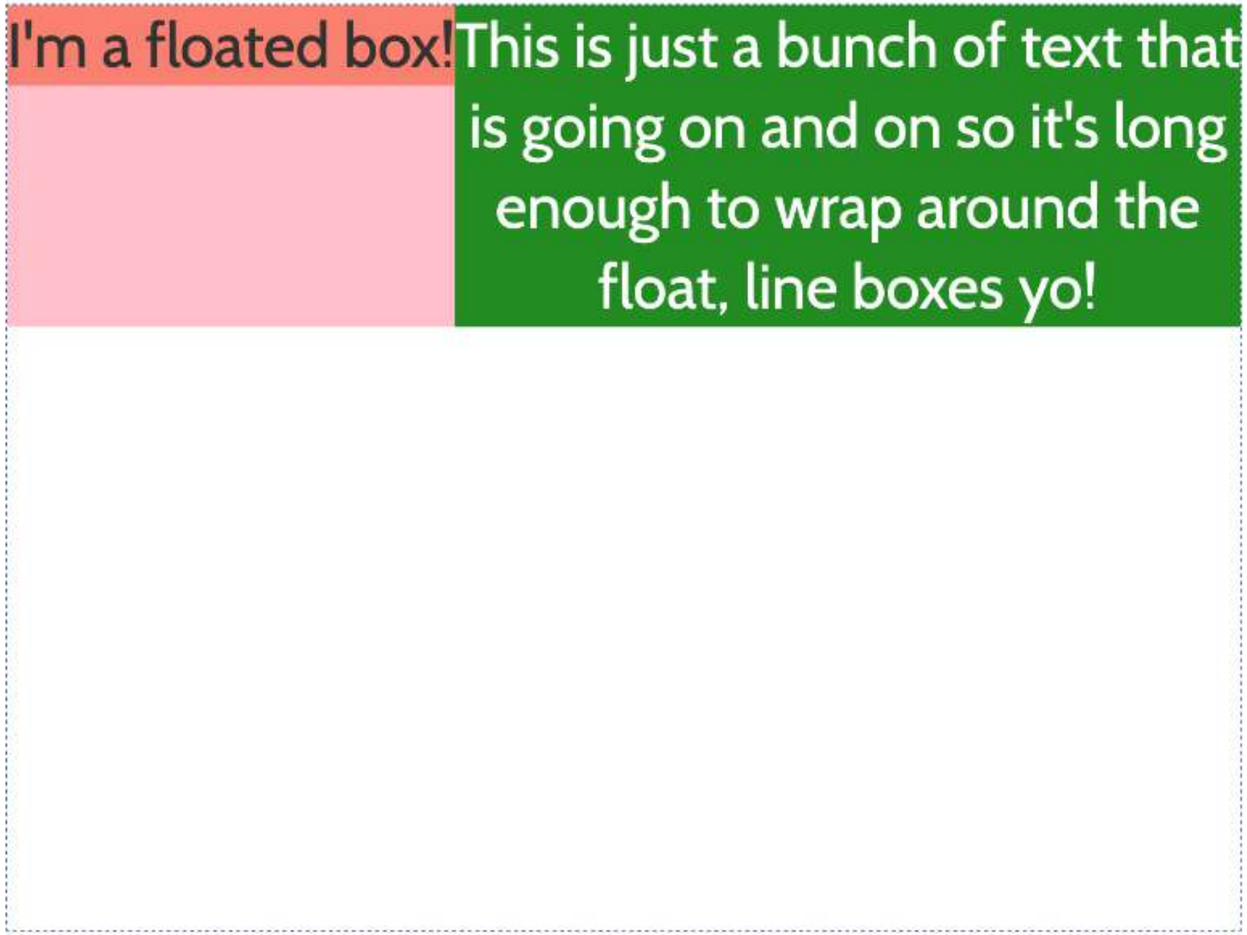
1. Prevent collapsing margins




```
<p>This is a line of text in a p tag.</
<div class="block-wrapper">
  <div class="box1">I'm a box with marg
  <div class="box2">I'm another box wi
</div>
```

```
.collapse .box2 {
  margin: 0.5em;
  display: inline-block;
}
```

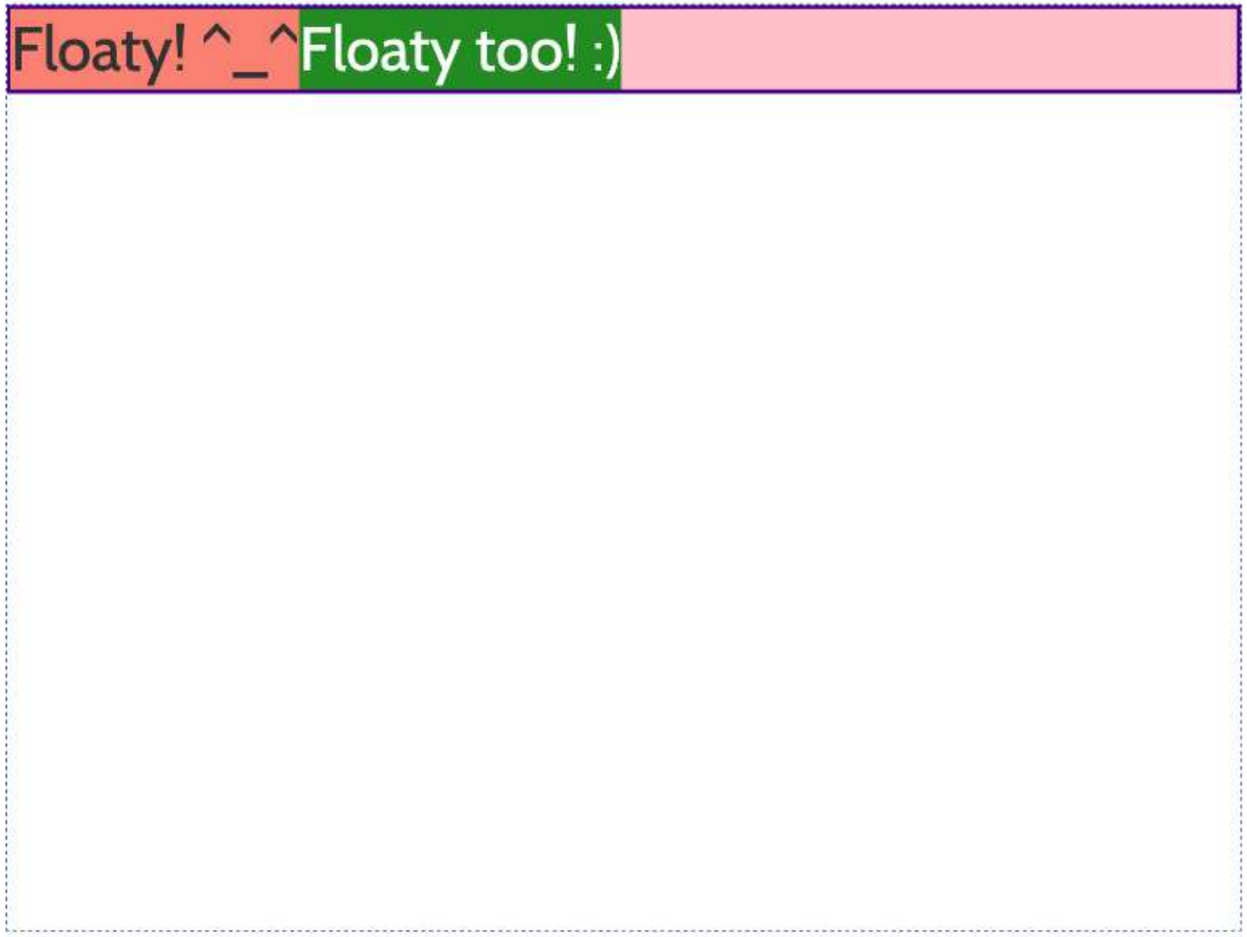
2. Stop text from flowing around the float




```
<div class="block-wrapper">
  <div class="box1">I'm a floated box!
  <p class="box2">This is just a bunch
</div>
```

```
.stop-flow .box1 {
  float: left;
}

.stop-flow .box2 {
  overflow: auto;
}
```

3. Contains floats

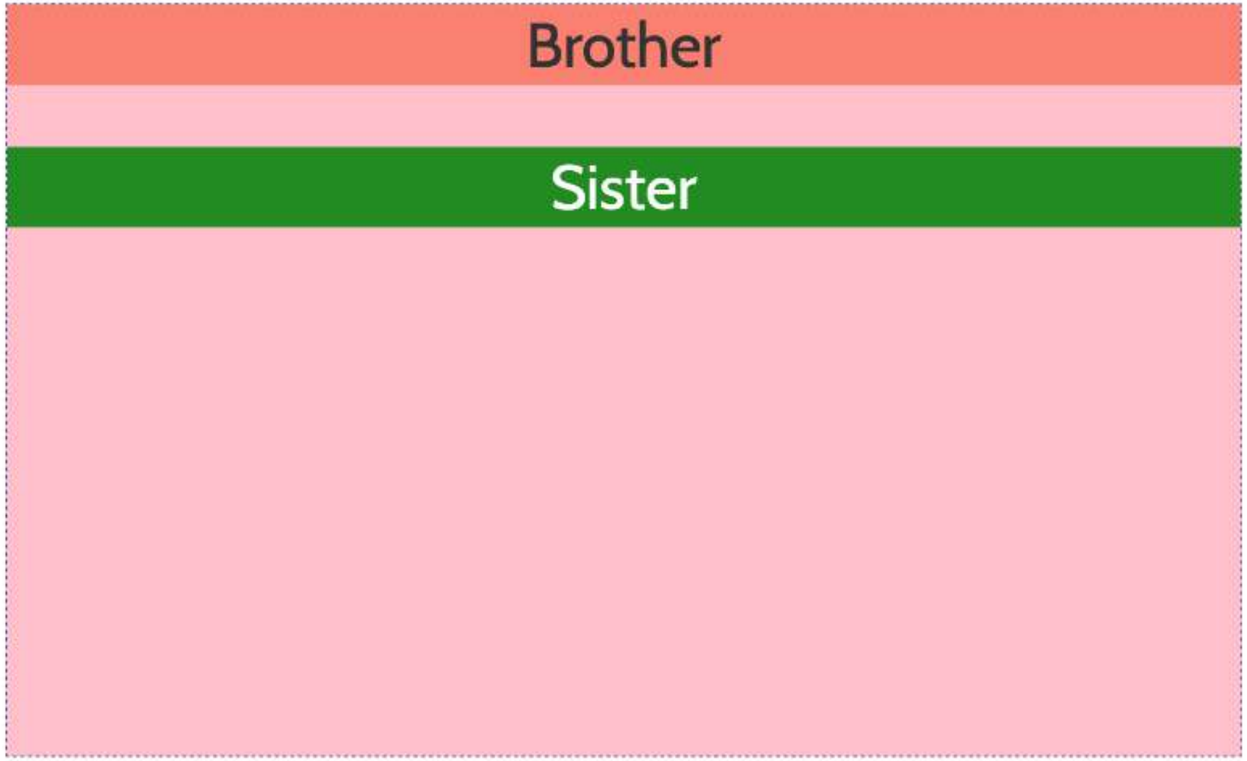



```html
<div class="block-wrapper">
  <p class="box1">Floaty! ^_^</p>
  <p class="box2">Floaty too! :)</p>
</div>
```

```css
.contain .block-wrapper {
  border: 3px solid indigo;
  display: flow-root;
}

.contain .box1 {
  float: left;
```

Let's talk about margin collapsing

1. Between adjacent siblings

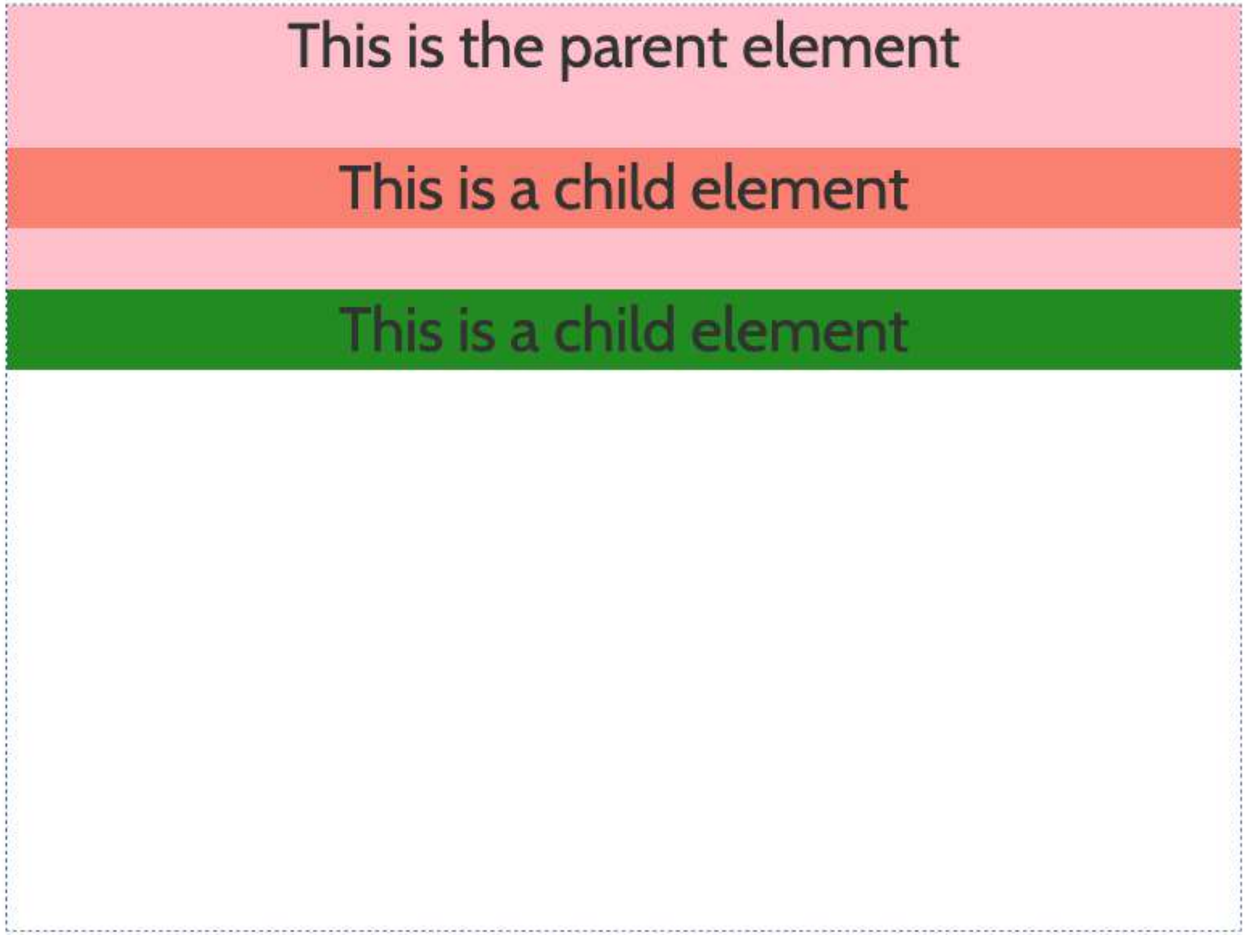



```
.siblings {
  writing-mode: horizontal-tb;
}

.siblings .brother {
  margin-bottom: 1em;
}

.siblings .sister {
  margin-top: 1em;
}
```

2. Between empty boxes


```
.empty {
  writing-mode: horizontal-tb;
}

.empty .nothing {
  margin-top: 1em;
  margin-bottom: 1em;
}
```

3. Parent and first / last child element



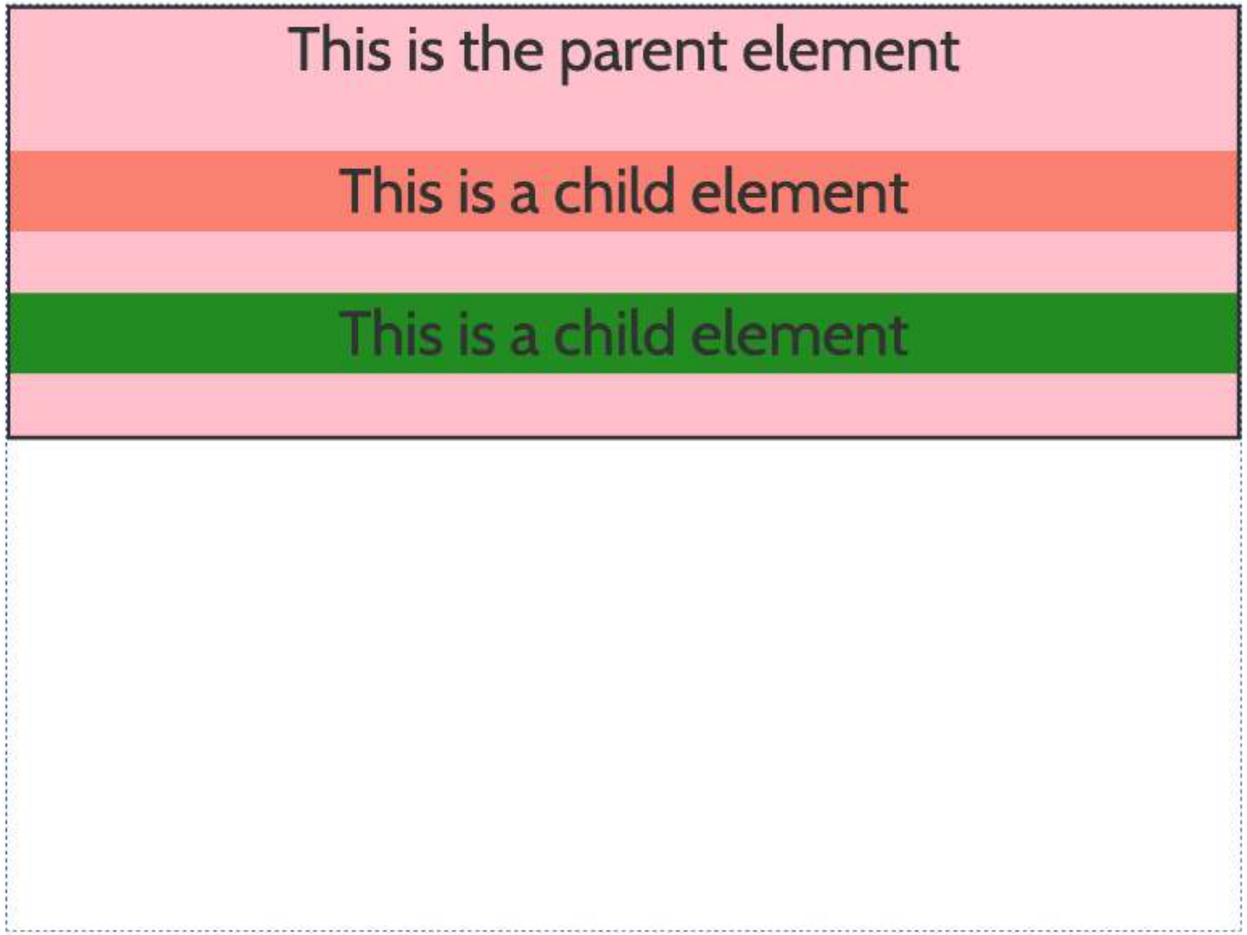

```
.family .parent {
  writing-mode: horizontal-tb;
}

.family .child {
  margin-bottom: 1em;
}
```

We prevent margin collapse by...?

1. Adding something in between the elements


```
.fixcollapse .nothing {
  margin-bottom: 1em;
  margin-top: 1em;
  padding: 0.009px;
}
```

2. Add border to the parent element




```
.fixcollapse .parent {
  border: 3px solid;
}

.fixcollapse .child {
  margin-bottom: 1em;
}
```

3. Create a new BFC

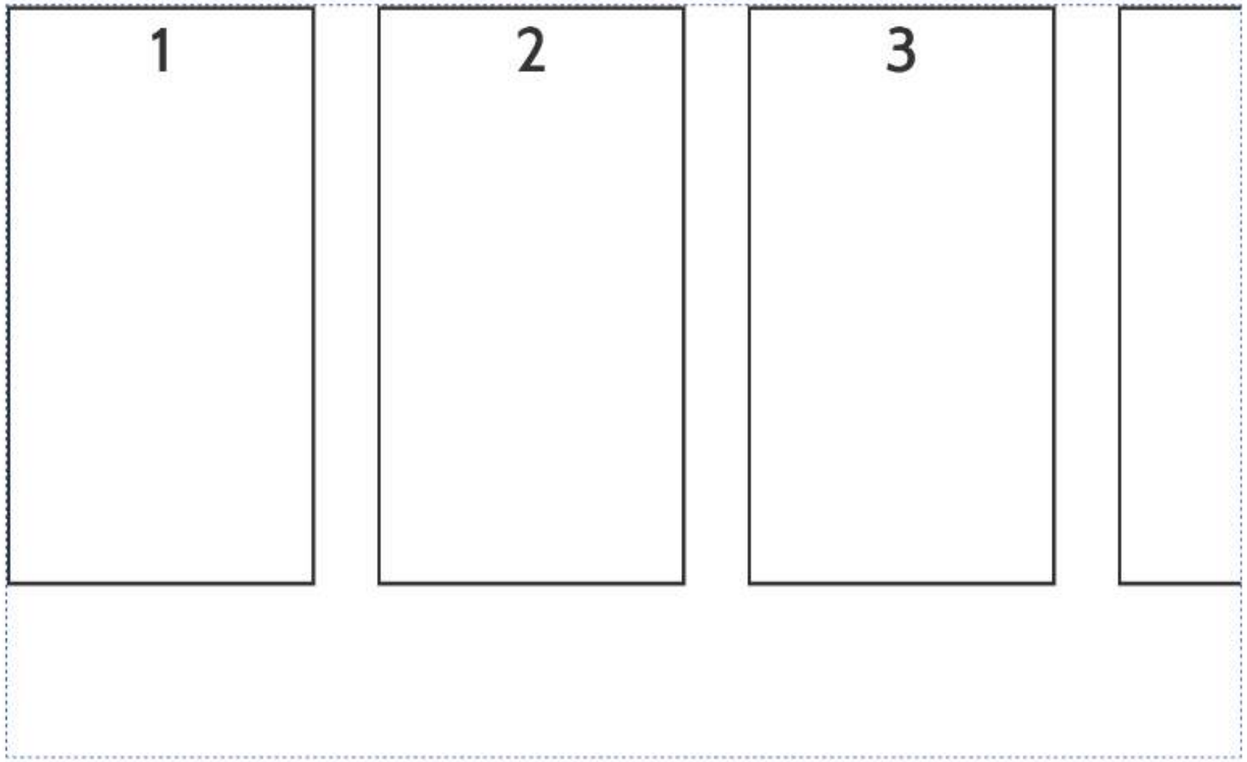
Refer to section on block formatting contexts ☝️

🤓 Everything You Need To Know About CSS Margins 🤓

https://www.smashingmagazine.com/2019/07/margins-in-css/

Grid gaps

May cause overflow if you're not careful




```
.gridgap .grid {
  display: grid;
  grid-template-columns: repeat(4,
25%);
  gap: 1em;
}
```

Overscroll and padding

padding at end side of overflow scroll container not applied

```
  overflow-x: scroll;
  padding: 1em;
}

.flexpad .flex__item {
  flex: 1 0 auto;
}

.flexpad .flex::after {
  content: '';
  padding-right: 1em;
}
```

Use the flex shorthand

> “Authors are encouraged to control flexibility using the flex shorthand rather than with its longhand properties directly, as the shorthand **correctly resets** any unspecified components to accommodate common uses.”

About shorthands...

The `border` shorthand

- Sets the same width, colour and style for all four borders of a box
- Unlike `margin` and `padding` shorthands, it **cannot** set different values on the four borders
- Also resets `border-image` to initial value

> It is therefore recommended that authors use the border shorthand, rather than other shorthands or the individual properties, to **override any border settings earlier** in the cascade.

The background shorthand

```
<bg-layer>#, <final-bg-layer>
```

where

```
<bg-layer> = <bg-image> || <bg-position> [ / <bg-size> ]? || <repeat-style> || <attachment> || <box> || <box>
```

```
<final-bg-layer> = <'background-color'> || <bg-image> || <bg-position> [ / <bg-size> ]? || <repeat-style> || <attachment> || <box> || <box>
```

- at least 1 value must occur, the rest is pretty much up to you
- for `<position>`, can optionally include `<bg-size>`
- entire set for `<bg-layer>` can occur multiple times, comma-separated
- only `<final-bg-layer>` can have `<'background-color'>`

> Given a valid declaration, for each layer the shorthand first sets the corresponding layer of each of `background-image`, `background-position`, `background-size`, `background-repeat`, `background-origin`, `background-clip` and `background-attachment` to that property's initial value, **then assigns any explicit values** specified for this layer in the declaration. Finally `background-color` is set to the specified color, if any, else set to its initial value.

The `animation` shorthand

```
<single-animation>#
```

where

```
<single-animation> = <time> || <easing-function> || <time> || <single-animation-iteration-count> || <single-animation-direction> || <single-animation-fill-mode> || <single-animation-play-state> || [ none | <keyframes-name> ]
```

Don't forget about the cascade

SCSS

Result

EDIT ON CODEPEN

```scss
@import url('https://fonts.googleapis.com/css?family=Lemon&display=swap');

body {
  display: flex;
  height: 100vh;
  font-family: 'Lemon', cursive;
}

div {
  display: grid;
  grid-template-columns: max-content max-content;
  gap: 1em;
  align-items: center;
  margin: auto;
  cursor: pointer;
}

.kittencorn {
  height: 65vmin;
  min-height: 12em;
}

.rainbow {
  fill: #a3e048;
}

p {
```

1×

Also, reading CSS specifications

CSS property syntax

Loosely based on the Backus-Naur Form (BNF) notation

A sandwich consists of a *lower slice of bread*, *mustard* or *mayonnaise*; optional *lettuce*, an optional slice of *tomato*; two to four slices of either *bologna*, *salami*, or *ham* (in any combination); one or more slices of *cheese*, and a *top slice of bread*.

```
sandwich ::= lower_slice [ mustard | mayonnaise ] lettuce? tomato? [
             bologna | salami | ham ]{2,4} cheese+ top_slice
```

Analogy from How to Read W3C Specs.

Slides | Cheatsheet

Thank you!

 https://www.chenhuijing.com

 @hj_chen

 @hj_chen

 @huijing

Header font is Biorhyme by Aoife Mooney
Body font is Cabin by Pablo Impallari