Advanced CSS

Creating efficient, modular and sustainable CSS solutions

adapt

Preprocessors

SASS/LESS/STYLUS

adapt

Why preprocessing and what is it?

- Building a function, reusing a definition or inheritance are hard to achieve.
- For bigger projects, or complex systems, maintenance is a very big problem.
- In some cases (Ex. background gradient, box-shadow, flexbox), coding with vendor specific properties become a burden. You have to add all different vendor versions for a single result.

Pre-processors, with their advanced features, helped to achieve writing reusable, maintainable and extensible codes in CSS. By using a pre-processor, you can easily increase your productivity, and decrease the amount of code you are writing in a project.

adapt

Declaring variables

Sass

```
$font-size: 16px;

div {
    font-size: $font-size;
}
```

Less

```
@font-size: 16px;

div {
    font-size: @font-size;
}
```

Stylus

```
font-size = 16px

div
    font-size font-size
```

Css

```
div {
    font-size: 16px;
}
```

https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_variables

adapt

Nestings

Sass

```sass
$link-color: #999;
$link-hover: #229ed3;

ul {
    margin: 0;

    li {
        float: left;
    }

    a {
        color: $link-color;

        &:hover {
            color: $link-hover;
        }
    }
}
```

Less

```less
@link-color: #999;
@link-hover: #229ed3;

ul {
    margin: 0;

    li {
        float: left;
    }

    a {
        color: @link-color;

        &:hover {
            color: @link-hover;
        }
    }
}
```

Stylus

```stylus
link-color = #999
link-hover = #229ed3

ul
    margin 0
    li
        float left
    a
        color link-color
        &:hover
            color link-hover
```

Css

```css
ul { margin: 0; }
ul li { float: left; }
ul a { color: #999; }
ul a:hover { color: #229ed3; }
```

adapt

Mixins

Sass

```
@mixin bordered($width) {
    border: $width solid #ddd;

    &:hover {
        border-color: #999;
    }
}

h1 {
    @include bordered(5px);
}
```

Less

```
.bordered (@width) {
    border: @width solid #ddd;

    &:hover {
        border-color: #999;
    }
}

h1 {
    .bordered(5px);
}
```

Stylus

```
bordered(w)
    border: n solid #ddd
    &:hover
        border-color: #999

h1
    bordered(5px)
```

Css

```
h1 { border: 5px solid #ddd; }
h1:hover { border-color: #999; }
```

adapt

Extend

Sass

```
.block { margin: 10px 5px; }

p {
  @extend .block;
  border: 1px solid #eee;
}

ul, ol {
  @extend .block;
  color: #333;
  text-transform: uppercase;
}
```

Less

```
.block { margin: 10px 5px; }

p {
  &:extend(.block);
  border: 1px solid #eee;
}

ul, ol {
  &:extend(.block);
  color: #333;
  text-transform: uppercase;
}
```

Stylus

```
.block
    margin 10px 5px

p
    @extend .block
    border 1px solid #eee

ul
ol
    @extend .block
    color #333
    text-transform uppercase
```

Css

```
.block, p, ul, ol { margin: 10px 5px; }

p { border: 1px solid #eee; }
ul, ol { color: #333; text-transform: upp
```

adapt

Color Operations

Sass

```
saturate($color, $amount)
desaturate($color, $amount)
lighten($color, $amount)
darken($color, $amount)
adjust-hue($color, $amount)
opacify($color, $amount)
transparentize($color, $amount)
mix($color1, $color2[, $amount])
grayscale($color)
complement($color)
```

Less

```
saturate(@color, @amount)
desaturate(@color, @amount)
lighten(@color, @amount)
darken(@color, @amount)
fadein(@color, @amount)
fadeout(@color, @amount)
fade(@color, @amount)
spin(@color, @amount)
mix(@color1, @color2, @weight)
grayscale(@color)
contrast(@color)
```

Stylus

```
red(color)
green(color)
blue(color)
alpha(color)
dark(color)
light(color)
hue(color)
saturation(color)
lightness(color)
```

adapt

IF/ELSE

Sass

```
@if lightness($color) > 30% {
    background-color: black;
}

@else {
    background-color: white;
}
```

Less

```
.mixin (@color) when (lightness(@color) >
    background-color: black;
}
.mixin (@color) when (lightness(@color) =
    background-color: white;
}
```

Stylus

```
if lightness(color) > 30%
    background-color black
else
    background-color white
```

adapt

Loops

Sass

```
@for $i from 1px to 3px {
    .border-#{i} {
        border: $i solid blue;
    }
}
```

Less

```
.loop(@counter) when (@counter > 0){
    .loop((@counter - 1));

    .border-@{counter} {
        border: 1px * @counter solid blue
    }
}
```

Stylus

```
for num in (1..3)
    .border-{num}
        border 1px * num solid blue
```

adapt

Math Operations

Sass

```
1cm * 1em => 1 cm * em
2in * 3in => 6 in * in
(1cm / 1em) * 4em => 4cm
2in + 3cm + 2pc => 3.514in
3in / 2in => 1.5
```

Less

```
1cm * 1em => 1cm * 1em
2in * 3in => 6in
(1cm / 1em) * 4em => 4cm
2in + 3cm + 2pc => 3.514in
3in / 2in => 1.5in
```

Stylus

```
1cm * 1em => 1 cm * em
2in * 3in => 6in
(1cm / 1em) * 4em => 4cm
2in + 3cm + 2pc => 5.181in
3in / 2in => 1.5in
```

Imports

Sass

```
@import "library";
@import "mixins/mixin.scss";
@import "reset.css";
```

Less

```
@import "library"
@import "mixins/mixin.less"
@import "reset.css"
```

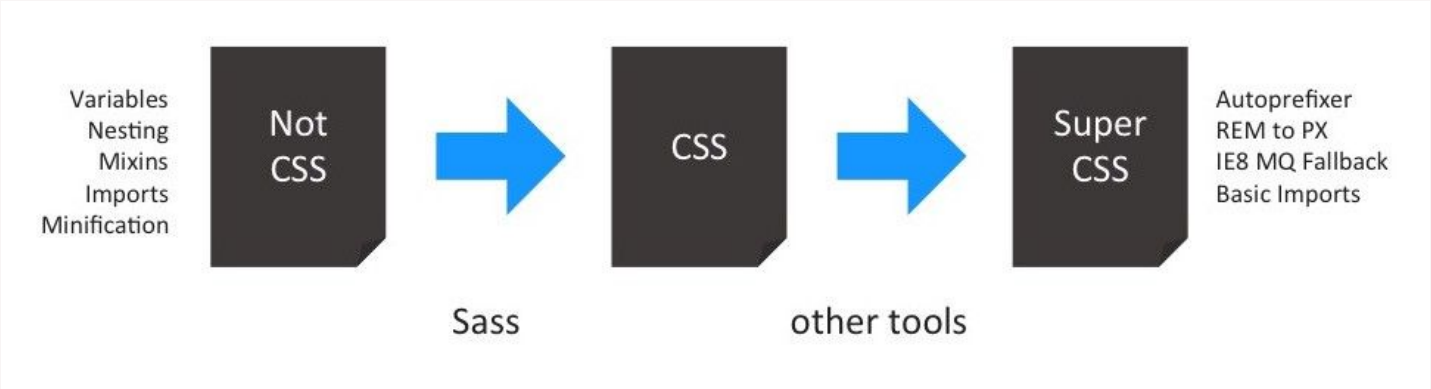
Stylus

```
@import "library"
@import "mixins/mixin.styl"
@import "reset.css
```

adapt

Post-processors

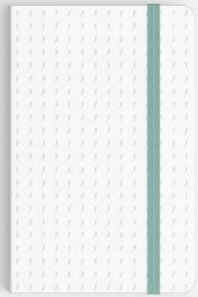
Post processors can do some heavy lifting for our compiled css : Add prefixes, convert rem to px values, make a fallback for IE8 media queries and other similar functionality. Post processors can also allow us to use polyfills for modern CSS4 functionality.



adapt

PRE-POST Processor reads

- Deconfusing post and preprocessing
- What Will Save Us from the Dark Side of CSS Pre-Processors?
- How Sass Can Save You a Lot of Time
- Sass Color Variables That Don't Suck by Landon Schropp
- Advanced SCSS, or 16 Cool Things You May Not Have Known Your Stylesheets Could Do by Jarno Rantanen
- Adjacent sibling selector specs by W3C
- Modular CSS naming conventions by John W. Long
- Responsive Web Design in Sass by Mason Wendell
- How to structure a Sass project by John W. Long



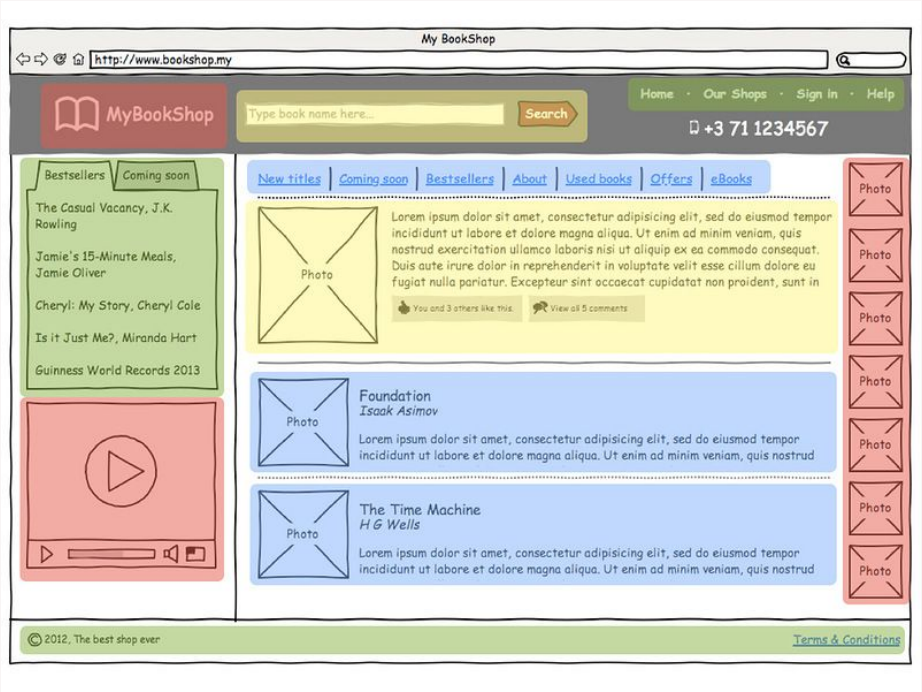
Writing more structured CSS

OOCSS, SMACSS and BEM

adapt

MODULAR CSS

Modular CSS is a collection of principles for writing code that is performant and maintainable at scale. It originated with developers at Yahoo and Yandex as a way to address the challenges of maintaining a large codebase. Some of the guidelines were controversial when introduced, but have since come to be recognized as best practices.



adapt

OOCSS

Object-Oriented CSS, or OOCSS, was created by Nicole Sullivan in 2009, and it's based on her work for Yahoo*. It's the origin point of Modular CSS. Her core concept was that objects are **reusable patterns** whose visual appearance is not determined by context.

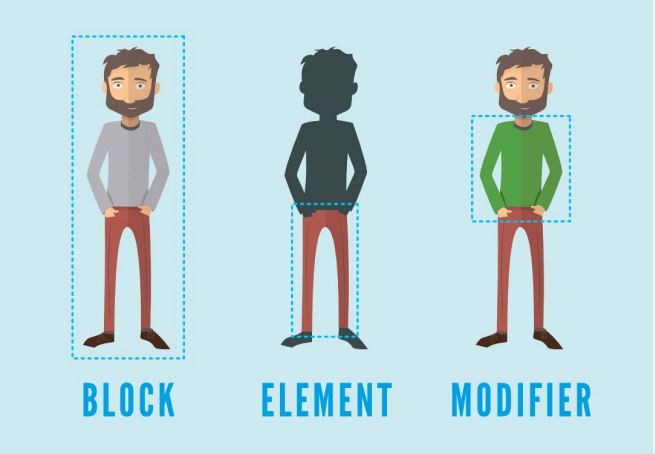
** For those of you who are going "Yahoo? Really?" You need to understand that their front-end team was doing some really cutting-edge stuff with the YUI library at the time. In 2009, Yahoo was not a dead-end tech company.*


"a CSS 'object' is a repeating visual **pattern**, that can be abstracted into an independent snippet of HTML, CSS, and possibly JavaScript. That object can then be **reused** throughout a site."
—Nicole Sullivan, @stubbornella
credit: John Morrison, goo.gl/AZBz7y

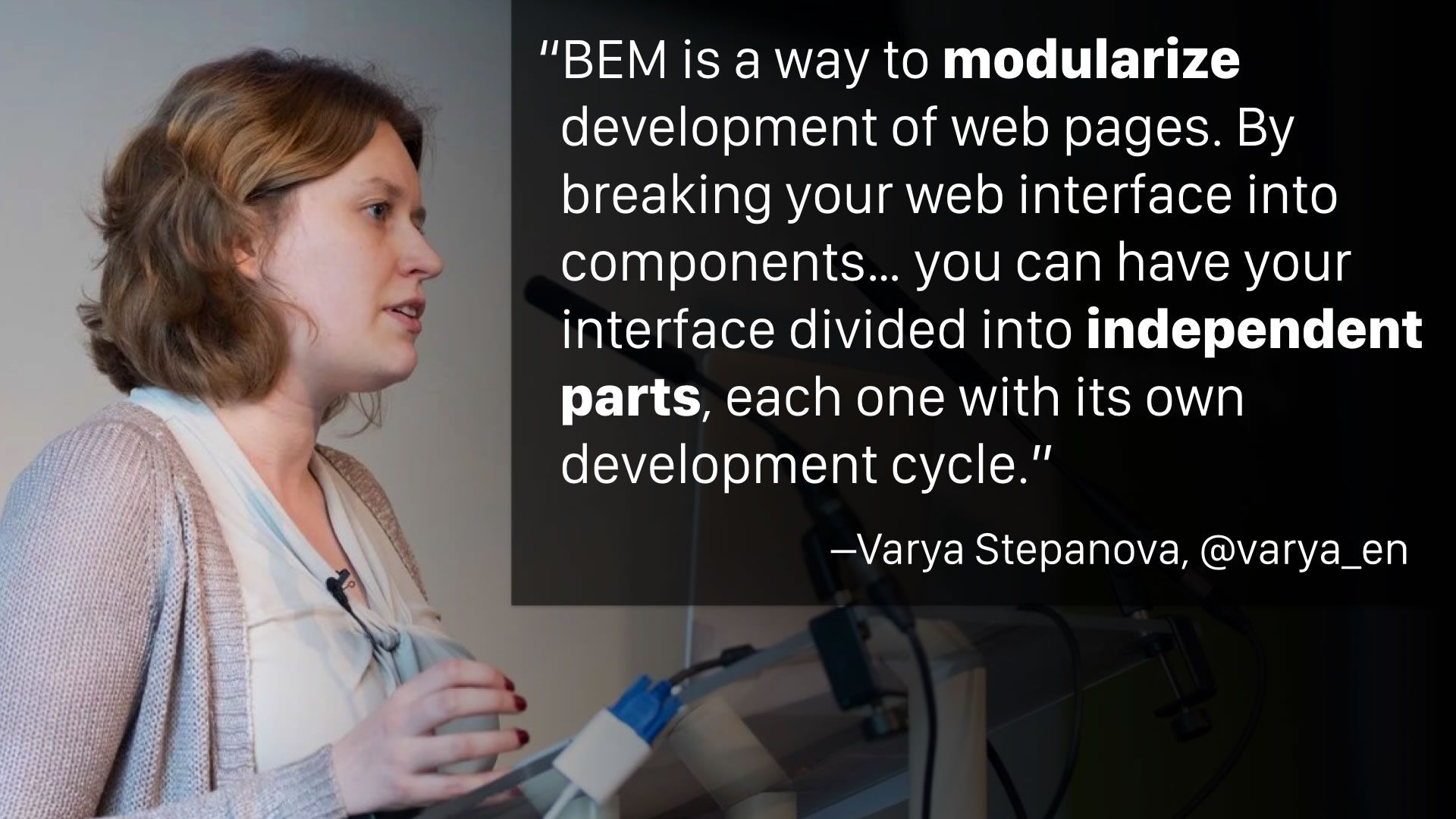
BEM - BLOCK, ELEMENT, MODIFIER

That brings us to the next big framework to help define the Modular CSS ethos. BEM, which stands for Block, Element, Modifier, was also created in 2009. It was developed at Yandex, which is like the Russian version of Google. They also operate a search engine and webmail program, so they were solving the same scale-related problems as Yahoo at the same time.

```
.block-name__element--modifier
```



adapt


"BEM is a way to **modularize** development of web pages. By breaking your web interface into components… you can have your interface divided into **independent parts**, each one with its own development cycle."
—Varya Stepanova, @varya_en

BLOCK

Blocks are nestable. They should be capable of being contained inside another block without breaking anything.

Blocks are repeatable. An interface should be able to contain multiple instances of the same block.

```
.card
```

```
.calendar
```

```
.login-form
```

ELEMENT (__)

Elements are the constituent parts of a block that can’t be used outside of it.

A good example is if you have a navigation menu, the items it contains don’t make sense outside the context of the menu. You wouldn’t define a block for a menu item. You’d have a block for the menu itself, and the menu items are child elements.

`.card__title` `.card__text` `.card__button`

`.calendar__month` `.calendar__day` `.calendar__row`

`.login-form__label` `.login-form__field` `.login__form__field-icon`

MODIFIER (--)

Modifiers define the appearance and behavior of a block.

For example, the appearance of the menu block may change from vertical to horizontal depending on the modifier that is used.

`.card__title--big` `.card__text--red` `.card__button--secondary`

`.calendar__month--current` `.calendar__day--selected`

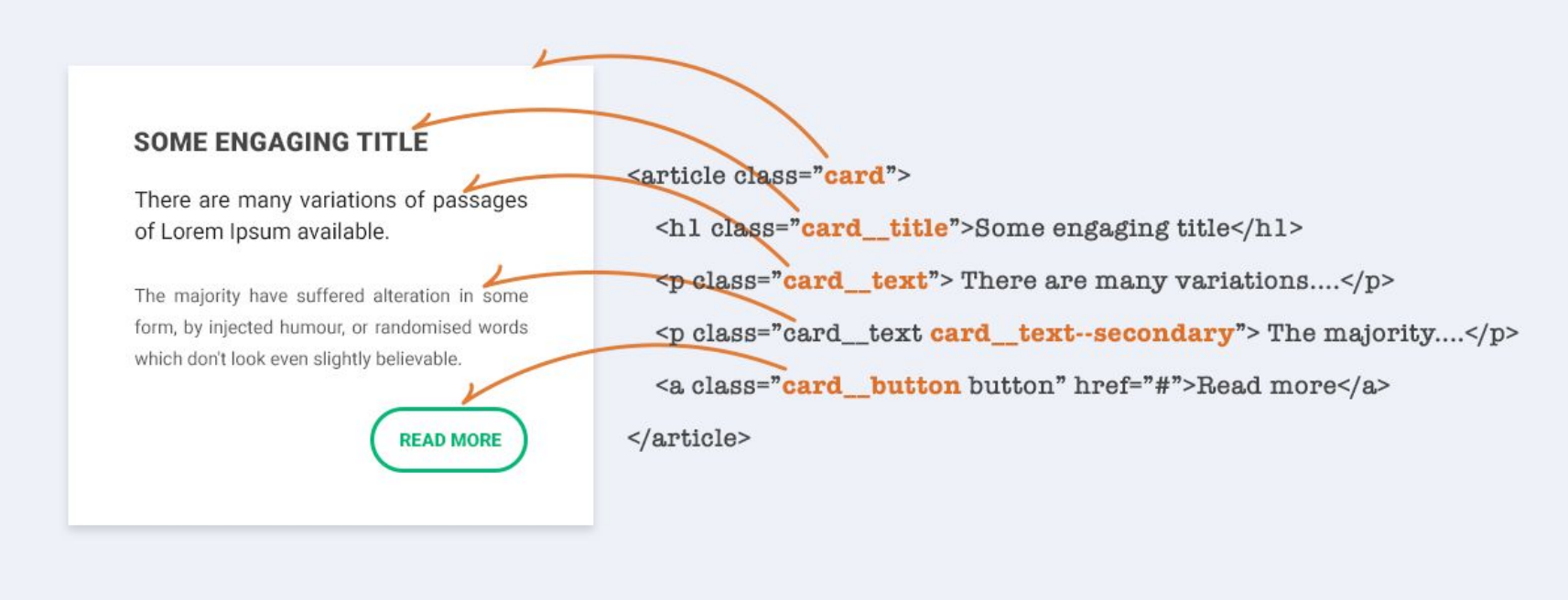
`.login-form--modal` `.login-form__field--error`

adapt

Reminder

- Names are written in **lower case**
- Words within names are separated by hyphens **(-)**
- Elements are delimited by double underscores **(__)**
- Modifiers are delimited by double hyphens **(--)**

BEM in example

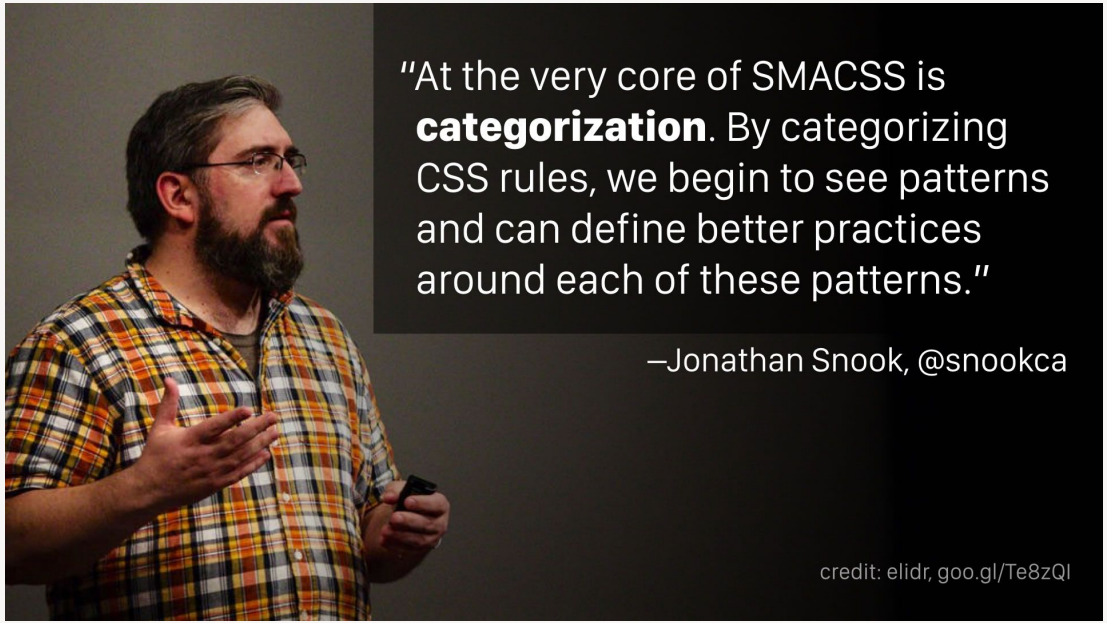


adapt

SMACSS

SMACSS, which stands for Scalable & Modular Architecture for CSS. It was created by Jonathan Snook in 2011. He also worked at Yahoo, writing CSS for Yahoo Mail.

The key concept that he added, building on OOCSS and BEM, was that different **categories** of components need to be handled differently.



adapt

Categories

1. **Base rules** are default styles for HTML elements like links, paragraphs, and headlines.

2. **Layout rules** divide the page into sections, and hold one or more modules together. They only define the layout, not color or typography.

3. **Modules (aka “objects” or “blocks”)** are the reusable, modular parts of a design. For example, buttons, media objects, product lists, etc.

4. **State rules** describe how modules or layouts look in a particular state. Typically applied or removed with JavaScript. For example, hidden, expanded, active, etc.

5. **Theme rules** describe how modules or layouts look when a theme is applied. For example, in Yahoo Mail, you might apply a user theme, which would affect every modules on the page. (This is really specific to apps like Yahoo. Most sites won’t use this category.)

Naming Convention Prefixes

The next principle he introduced was using prefixes to differentiate between categories of rules. He liked the idea that BEM had a clear naming convention, but he wanted to be able to tell at a glance what type of module he was looking at.

- `l-` is used as a prefix for layout rules: `l-inline`
- `m-` is used as a prefix for module rules: `m-callout`
- `is-` is used as a prefix for state rules: `is-collapsed`

Breakdown

Styles in a modular system can be organized into the following categories:

- **Base rules** are default styles for HTML elements. **Examples**: `a, li, h1`
- **Layout rules** control how modules are laid out, but not visual appearance. **Examples**: `.l-centered, .l-grid, .l-fixed-top`
- **Modules** are visual styles for reusable, self-contained UI components. **Examples:** `.m-profile, .m-card, .m-modal`
- **State rules** are added by JavaScript. **Examples:** `.is-hidden, .is-collapsed, .is-active`
- **Helper** (aka utility) rules are small in scope and independent of modules. Examples: .h-uppercase, .h-nowrap, .h-muted

Modular CSS rules

When writing styles in a modular system, follow these rules:

- Don’t use IDs
- Don’t nest CSS deeper than one level
- Add classes to child elements
- Follow a naming convention
- Prefix class names

All together

Bad

```
<div class="box profile pro-user">
  <img class="avatar image" />
  <p class="bio">...</p>
</div>
```

Good

```
<div class="l-box m-profile m-profile--is-pro-user">
  <img class="m-avatar m-profile__image" />
  <p class="m-profile__bio">...</p>
</div>
```

adapt

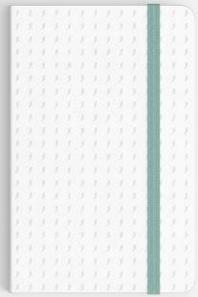
Modular CSS reads

- 8 rules for a robust, scalable CSS architecture, by Jarno Rantanen (10/16/2016)
- More Meaningful CSS, by Jonathan Snook (5/17/16)
- CSS and Scalability, by Adam Morse (3/24/16)
- Can CSS Be Too Modular?, by Harry Roberts (3/8/15)
- Side Effects in CSS, by Philip Walton (3/3/15)
- Used and Abused -- CSS Inheritance and Our Misuse of the Cascade, by Micah Godbolt (8/25/14)
- Enduring CSS: writing style sheets for rapidly changing, long-lived projects, by Ben Frain (8/7/14)
- Challenging CSS Best Practices -- suggesting a unique approach (all utility classes) the author calls Atomic CSS -- by Thierry Koblentz (10/21/13)
- Atomic Design, by Brad Frost (6/10/13) -- which isn't really about *CSS code*, exactly; but so many people have found it a valuable way to think about frontend component architecture that it fits in well with this list.
- MindBEMding - getting your head around BEM syntax, by Harry Roberts (1/25/13)
- CSS Architecture, by Philip Walton (11/16/12)
- SOLID CSS, by Miller Medeiros (9/10/12)
- Shoot to kill: CSS selector intent, by Harry Roberts (7/17/12)
- The open/closed principle applied to CSS, by Harry Roberts (6/21/12)
- Keep your CSS selectors short, by Harry Roberts (5/15/12)
- The single responsibility principle applied to CSS, by Harry Roberts (4/28/12)
- About HTML semantics and frontend architecture, by Nicolas Gallagher (3/15/12)
- An Introduction to Object Oriented CSS (OOCSS), by Louis Lazaris (12/12/11)

adapt

Transitions

Changing the value of a property from one state to another by using certain parameters



adapt

Property structure

```
.container {
  transition: property
              duration
              timing-function
              delay;
}

a {
  color: red;
  font-size: 14px;
  transition: color 0.2s linear;
}

a:hover {
  color: blue;
}
```

Property	Description
`transition-property`	the CSS property that should transition
`transition-duration`	the duration of the transition
`transition-timing-function`	the timing function used by the animation (common values: linear, ease). Default: ease
`transition-delay`	optional number of seconds to wait before starting the animation

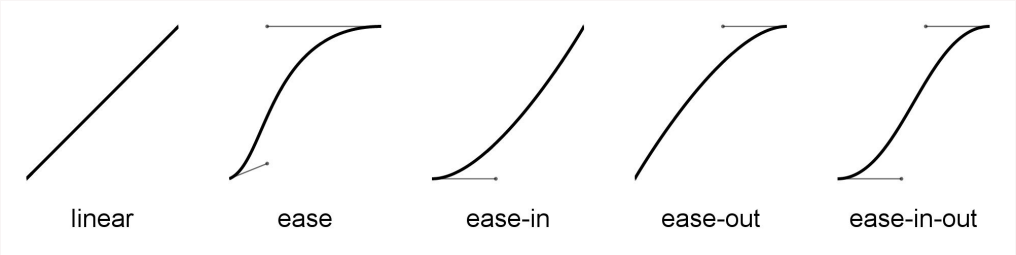
Properties that can be animated

- background-color
- background-position
- border-color
- border-width
- border-spacing
- bottom
- clip
- color
- crop
- font-size
- font-weight
- height
- left
- letter-spacing
- line-height
- margin
- max-height
- max-width
- min-height
- min-width
- opacity
- outline-color
- outline-offset
- outline-width
- padding
- right
- text-indent
- text-shadow
- top
- vertical-align
- visibility
- width
- word-spacing
- z-index

adapt

Timing functions

```
/* Keyword values */
transition-timing-function: ease;
transition-timing-function: ease-in;
transition-timing-function: ease-out;
transition-timing-function: ease-in-out;
transition-timing-function: linear;
transition-timing-function: step-start;
transition-timing-function: step-end;

/* Function values */
transition-timing-function: steps(4, end);
transition-timing-function: cubic-bezier(0.1, 0.7, 1.0, 0.1);
transition-timing-function: frames(10);

/* Multiple timing functions */
transition-timing-function: ease, step-start, cubic-bezier(0.1, 0.7, 1.0, 0.1);
```



Cubic-bezier builder: http://www.roblaplaca.com/examples/bezierBuilder/

adapt

Keyframes - way to describe an animation

```
@keyframes identifier {
  0% { top: 0; }
  50% { top: 30px; left: 20px; }
  50% { top: 10px; }
  100% { top: 0; }
}
```

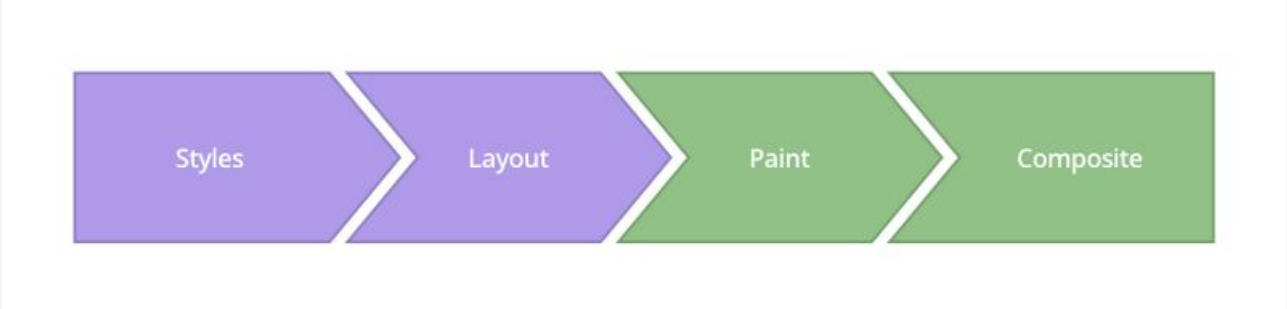
```
@keyframes slidein {
  from {
    margin-left: 100%;
    width: 300%;
  }

  to {
    margin-left: 0%;
    width: 100%;
  }
}
```

```
/* @keyframes duration |
timing-function | delay |
iteration-count | direction |
fill-mode | play-state | name */
animation: 3s ease-in 1s 2
reverse both paused slidein;
```

adapt

Be aware of Critical Rendering Path.

1. **Styles.** The browser starts calculating the styles to apply in elements.
2. **Layout.** This is where the browser sets the page properties such as **width and height, as well as its margins, or left/top/right/bottom** for instance.
3. The browser will now start filling in the pixels for each element into layers. The properties that it uses are, for instance: **box-shadow, border-radius, color, background-color, among others.**
4. **Composite.** Browser starts drawing layers on screen. Modern browsers can animate four styles pretty, pretty well, making use of the **transform and opacity** properties.



adapt

Transitions reads

- “The Story of CSS Grid, from Its Creators,” Aaron Gustafson, A List Apart
- “Internet Explorer and Edge Virtual Machines for Testing,” Microsoft
- “BrowserStack,” A cross-browser testing tool
- “Should I try to use the IE implementation of Grid Layout?” Rachel Andrew
- The Anti-hero of CSS Layout — “display:table”,” Colin Toh
- “CSS Grid Fallbacks and Overrides cheatsheet” Rachel Andrew
- “Using Feature Queries in CSS,” Jen Simmons, Mozilla Hacks
- “A video tutorial on Feature Queries,” Rachel Andrew
- “CSS Grid and Progressive Enhancement,” MDN web docs, Mozilla

Responsive WD

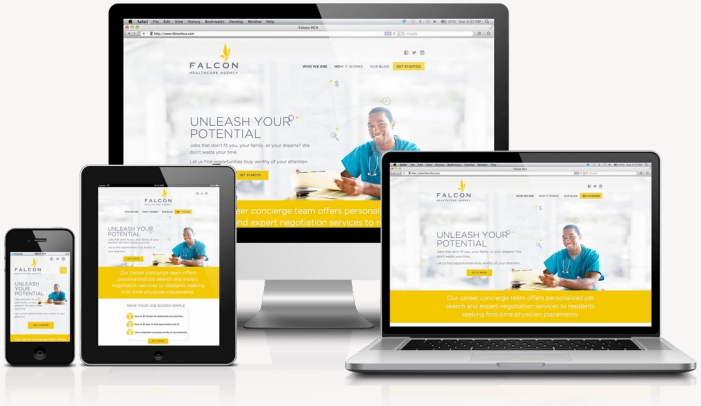
Breakpoints, units and logic

adapt

Breaking it down

Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation.

RWD is broken down into three main components: flexible layouts, media queries, and flexible media.



adapt

Flexible layouts

Building a flexible grid, capable of dynamically resizing to any width.

This is achievable by using relative units (%, rem, em) instead of fixed units such as pixels or inches. Most common properties to use this are `margin, padding, width, height, border.`

The formula for getting the perfect relative unit :

`target ÷ context = result`

```
300px % 960px = 0.3125 * 100 = 31.25%
32px % 16px = 2rem
```

```
section,
aside {
  margin: 1.858736059%; /*  10px ÷ 538px =
.018587361 */
}
section {
  float: left;
  width: 63.197026%;    /* 340px ÷ 538px =
.63197026 */
}
aside {
  float: right;
  width: 29.3680297%;  /* 158px ÷ 538px =
.293680297 */
}
```

adapt

EM vs REM

REM and em units are computed into pixel values by the browser, based on font sizes in your design. Don’t use em or rem in multi column layout widths - use % instead. Don’t use em or rem if scaling would unavoidably cause a layout element to break.	
em units are based on the font size of the element they’re used on.	rem units are based on the font size of the html element.
em units can be influenced by font size inheritance from any parent element	rem units can be influenced by font size inheritance from browser font settings.
Use em units for sizing that should scale depending on the font size of an element other than the root.	Use rem units for sizing that doesn’t need em units, and that should scale depending on browser font size settings.
Use em units on media queries	Use rem units unless you’re sure you need em units, including on font sizes.

adapt

Media Queries

Media queries provide the ability to specify different styles for individual browser and device in certain circumstances

SINTAXIS

```
@media not|only mediatype and|not|only
(media feature) {
  .my-code { ... }
}
```

MEDIATYPE

```
@media not|only mediatype
all
print
screen
speech
braile / projection / tv (Deprecated)
```

MEDIA FEATURE

```
@media and|not|only (media feature)
orientation (landscape / portrait)
max resolution / min resolution
device-pixel-ratio
caniuse.com/#feat=css-media-resolution
Más:w3.org/TR/css3-mediaqueries/#media1
```

adapt

Use nuances

Can be used when embedding stylesheets in order to load only specific css for different devices.

```
<link rel="stylesheet" type="text/css" href="print.css"  media="print">
```

Can be used in CSS

```
@media only screen and (min-width: 599px) { ... }
@media screen and (min-width: 600px) and (max-width: 800px) { ... }
@media all and (min-width: 1280px) not (aspect-ratio: 16/9) { ... }
@media only screen and (landscape: portrait) { ... }
```

Add this line to </head> element to prevent zoom when user open the application

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

In addition, we can disable zoom completely with **`user-scalable=false`.**

https://css-tricks.com/snippets/css/media-queries-for-standard-devices/
https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

adapt

Progressive enhancement vs. Graceful degradation

Progressive Enhancement - Mobile first	**Graceful Degradation - Desktop first**
Progressive enhancement methodology is a guiding principle with the aim of making web pages accessible to a variety of browsers, by developing from a baseline of compatible features, and then layering enhanced features throughout progression to more capable browsers. **Basic functionality > Advanced functionality**	Graceful degradation methodology is a guiding principle with the aim of making web pages accessible to a variety of browsers, by developing from a baseline of full features, and then removing layers of features throughout the regression to less capable browsers **Advanced functionality > Basic functionality**

adapt

Flexible Embedded Media

As viewports begin to change size media doesn't always follow suit. Images, videos, and other media types need to be scalable, changing their size as the size of the viewport changes.

`Iframes` are particularly bad when resizing and max-width does not work as expected. Luckily there is a fix for it which helps to maintain proportion.

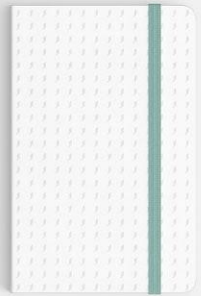
```
<figure>
  <iframe
src="https://www.youtube.com/embed/4Fqg43ozz7A"></iframe>
</figure>
```

```
img, video, canvas {
  max-width: 100%;
  height: auto; // it depends
  width: 100%; // it depends
}

figure {
  height: 0;
  padding-bottom: 56.25%; /* 16:9 */
  position: relative;
  width: 100%;
}

iframe {
  height: 100%;
  left: 0;
  position: absolute;
  top: 0;
  width: 100%;
}
```

adapt

RWD reads

- What We Mean When We Say “responsive”
- The State of Responsive Web Design
- Responsive Typography: The Basics
- Responsive Web Design (RWD) and User Experience
- 10 Responsive Design Problems and Fixes
- The Five Golden Rules of Responsive Web Design
- You May Be Losing Users If Responsive Web Design Is Your Only Mobile Strategy
- The Next Big Thing In Responsive Design
- RWD Bloat



ADVANCED LAYOUTS

FLEX & GRID

adapt

FLEX & GRID

CSS3 introduced two new layout methods as an alternative to abusing floats and tables:

- **Grid layout** divides space into columns & rows. Like table layouts, but better! The Grid layout spec is still being developed, and isn't ready for use yet.
- **Flexbox layout** distributes space along a single column or row. Like float layouts, but better! The Flexbox spec is finished and enjoys excellent browser support today.

The previous version of CSS defined four layout modes:

- **block layout** for laying out documents
- **inline layout** for laying out text
- **table layout** for laying 2D tabular data
- **positioned layout** for explicit positioning

adapt

FLEXBOX

Flexbox is a layout mode added in CSS3 to replace hacky float and table layouts. It comes with a clever set of defaults that make it easy to create complex layouts with relatively little code. It's stable, easy-to-use, and well-supported.

Syntax

```
.flex-container {
  display: -webkit-box;
  display: -moz-box;
  display: -ms-flexbox;
  display: -webkit-flex;
  display: flex;
}
```

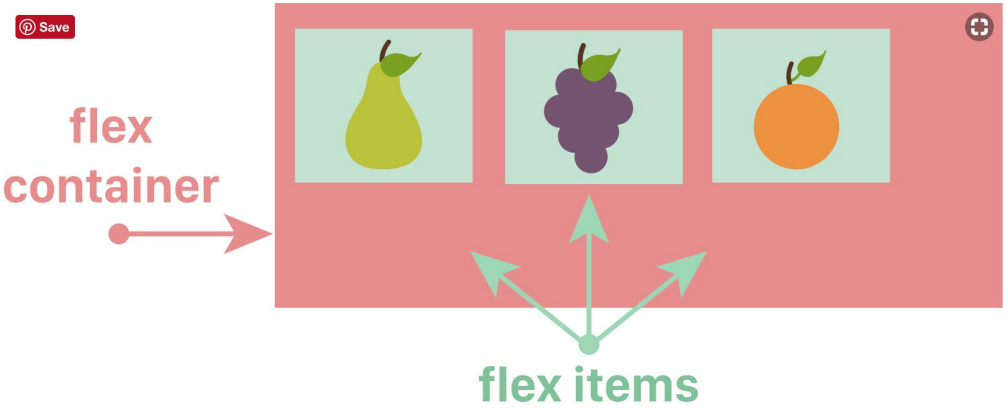


Image credit: Flexbox Cheatsheet

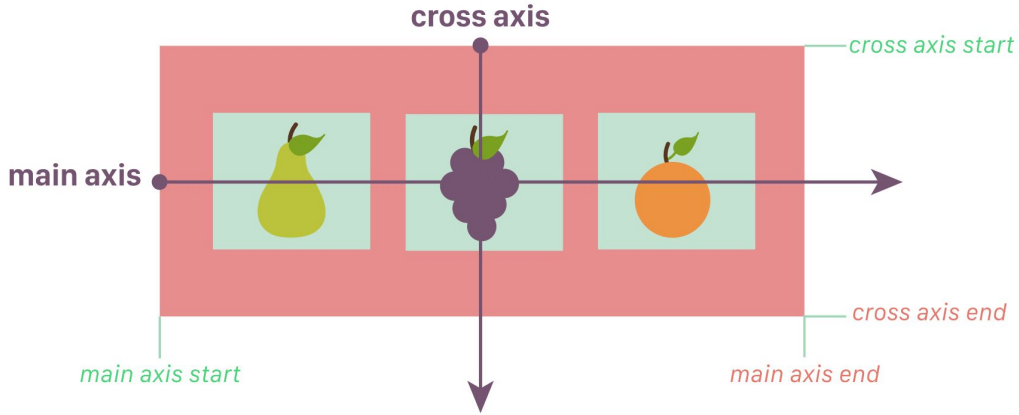


Image credit: Flexbox Cheatsheet

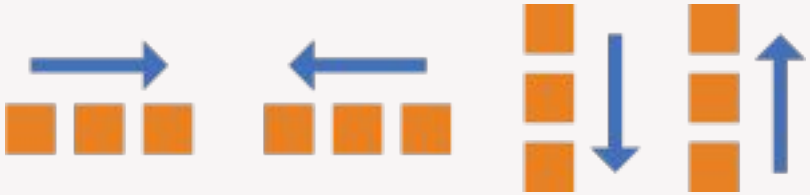
display: flex; does all the following:

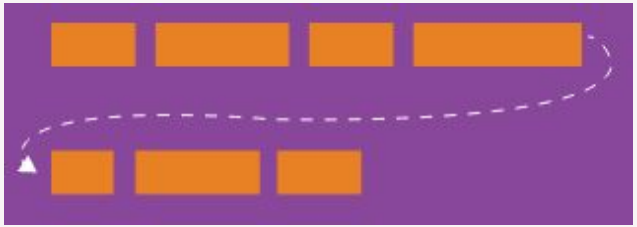
1. Treats `.flex-container` as a flex container.
2. Treats all direct children of `.flex-container` as flex items.
3. Flex items will be laid out in a horizontal line.
4. Flex items will be laid out in source order.
5. Flex items will be laid out starting from the left side of the flex container.
6. Flex items will be sized based on their regular `width` properties.
7. If there’s not enough space for all the flex items, they will be allowed to shrink horizontally until they all fit.
8. If they need to shrink, each item will shrink equally.
9. Flex items will all stretch vertically to match the height of the tallest flex item.

Flex-direction & flex-wrap = flex-flow

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.

```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
  flex-wrap: nowrap | wrap | wrap-reverse;
  flex-flow: <‘flex-direction’> || <‘flex-wrap’>
}
```



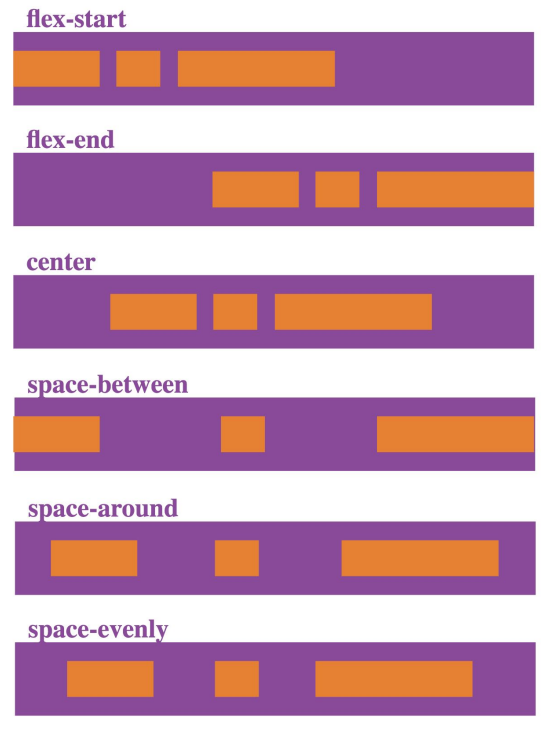


adapt

Justify content

This defines the alignment along the main axis. It helps distribute extra free space left over 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.

```
.container {
  justify-content: flex-start | flex-end | center | space-between |
space-around | space-evenly;
}
```

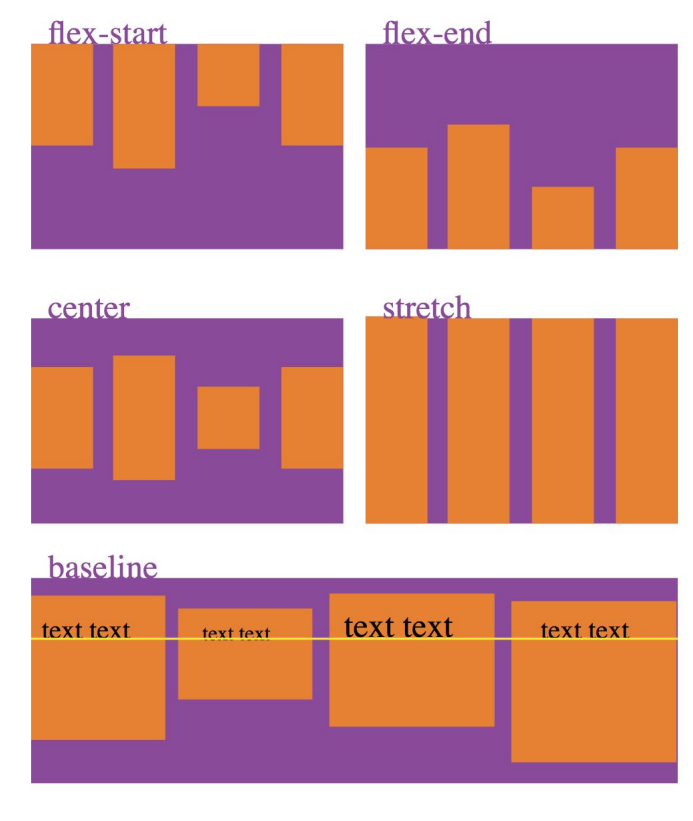


adapt

Align items

This defines the default behaviour for how flex items are laid out along the cross axis on the current line. Think of it as the justify-content version for the cross-axis (perpendicular to the main-axis).

```
.container {
  align-items: flex-start | flex-end | center |
baseline | stretch;
}
```



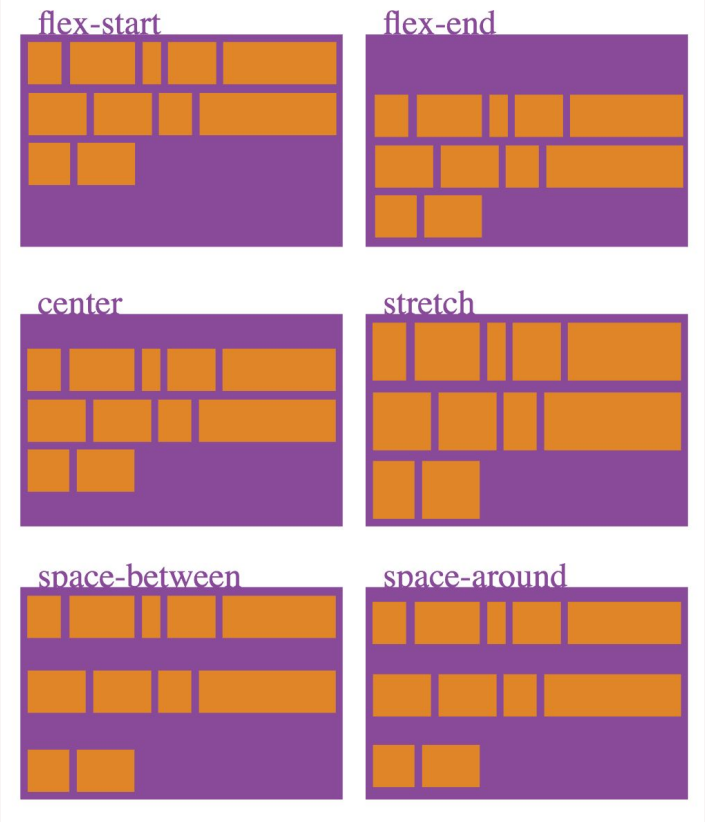
adapt

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.

Note: this property has no effect when there is only one line of flex items.

```
.container {
  align-content: flex-start | flex-end | center |
space-between | space-around | stretch;
}
```



adapt

Flex children can have properties too

Each children can override it's cross axis positioning and change how it resizes

```
.item {
  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;
}
```

Ex.

```
.item {
  order: 2;
  flex-basis: 20%;
  flex: 2 1 auto;
  align-self: center;
}
```

adapt

Flexbox reads

1. Flexbox playground
2. A guide to flexbox
3. Understanding flexbox - everything you need to know

What CSS Grids Are (And Why You Should Use Them)

There are plenty of reasons for this, but the most common is that **it's simpler than the potentially frustrating methods of using floats and positioning to design layouts.** There's usually some degree of incompatibility between browsers, after all, meaning that floats you set or elements you position won't always appear the same way. The path of least resistance, therefore, is to create simple, easily-portable layouts that render similarly regardless of the device or browser being used.

```
<div class="blog-layout">

  <div class="header">Header</div>

  <div class="content">Content</div>

  <div class="sidebar">Sidebar</div>

  <div class="footer">Footer</div>

</div>
```

```
.blog-layout {
  display: grid;
  grid-template-columns: 400px 20px 180px;
  grid-template-rows: 100px 20px 210px 20px 100px;
}
```

adapt

ALL | FRONTEND | BACKEND

JENN SCHIFFER
FOG CREEK

Keynote

JOHN GRAHAM-CUMMING
CLOUDFLARE

The New Reality of DDoS

JAMES BURNS
TWILIO

Practical Distributed Systems: Building for the real world with distributed tracing and chaos engineering

KARISSA MCKELVEY
CODE FOR SCIENCE AND SOCIETY

Keynote

NIKITA BAKSALYAR
MAIDSAFE.NET

Exploring the world of decentralized networks with WebRTC

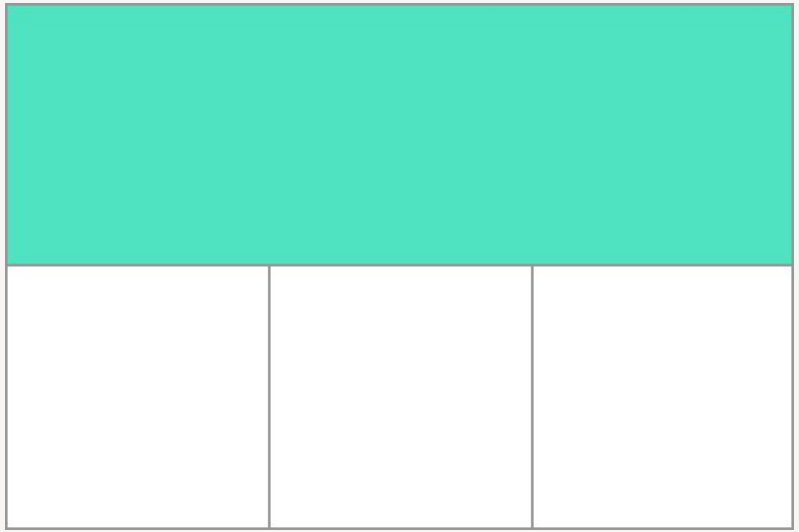
SARAH DRASNER

SVG can do THAT?!

Grid Tracks

We define rows and columns on our grid with the `grid-template-columns` and `grid-template-rows` properties. These define grid tracks. A grid track is the space between any two lines on the grid.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
  grid-template-rows: 200px 200px;
  grid-auto-rows: 200px;
}
```



adapt

FR unit, auto-rows, repeat() and minmax()

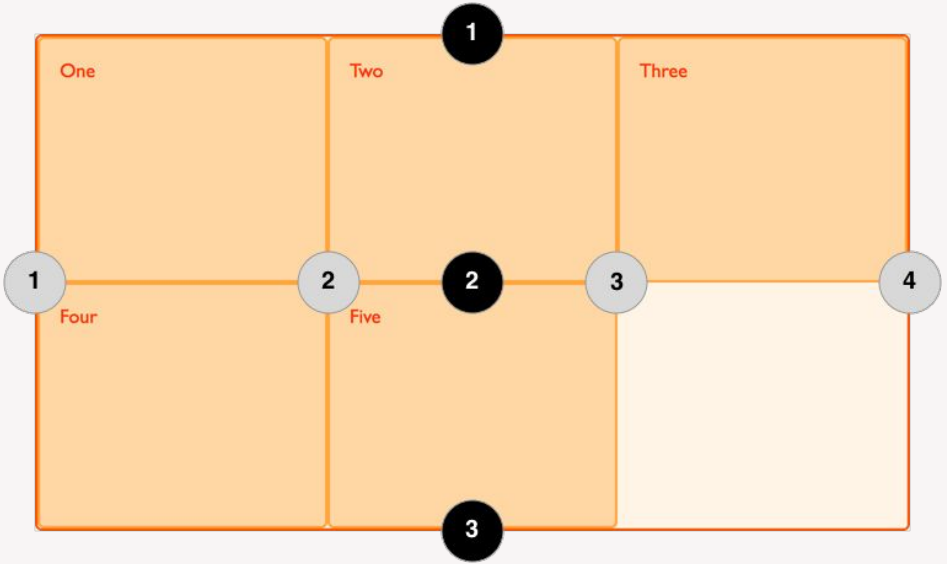
1. Large grids with many tracks can use the repeat() notation, to repeat all or a section of the track listing.
2. Tracks can be defined using any length unit. Grid also introduces an additional length unit to help us create flexible grid tracks. The new fr unit represents a fraction of the available space in the grid container.
3. You can also define a set size for tracks created in the implicit grid with the grid-auto-rows and grid-auto-columns properties.
4. When setting up an explicit grid or defining the sizing for automatically created rows or columns we may want to give tracks a minimum size, but also ensure they expand to fit any content that is added. Grid has a solution for this with the minmax() function.

```
.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-columns: repeat(3, 1fr);    grid-template-columns: 1fr repeat(2, 1fr);
  grid-auto-columns: 1fr;
  grid-auto-rows: 200px;
  grid-auto-rows: minmax(100px, auto);
}
```

adapt

Positioning items against lines

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 100px;
}

.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}

.box2 {
  grid-column-start: 1;
  grid-row-start: 3;
  grid-row-end: 5;
}
```



adapt

@supports

The **@supports** CSS at-rule lets you specify declarations that depend on a browser's support for one or more specific CSS features. This is called a *feature query*. The rule may be placed at the top level of your code or nested inside any other conditional group at-rule.

Syntax:

```
@supports <supports-condition> {
  <group-rule-body>
}
```

```
@supports (display: grid) {
  div {
    display: grid;
  }
}

@supports not (display: grid) {
  div {
    float: right;
  }
}

@supports not (not (transform-origin: 2px))
@supports (display: grid) and (not (display:
inline-grid))
```

adapt

Grid reads

- “The Story of CSS Grid, from Its Creators,” Aaron Gustafson, A List Apart
- “Internet Explorer and Edge Virtual Machines for Testing,” Microsoft
- “BrowserStack,” A cross-browser testing tool
- “Should I try to use the IE implementation of Grid Layout?” Rachel Andrew
- The Anti-hero of CSS Layout — “display:table”,” Colin Toh
- “CSS Grid Fallbacks and Overrides cheatsheet” Rachel Andrew
- “Using Feature Queries in CSS,” Jen Simmons, Mozilla Hacks
- “A video tutorial on Feature Queries,” Rachel Andrew
- “CSS Grid and Progressive Enhancement,” MDN web docs, Mozilla

Lets get physical

Make sure you have and use latest node/npm

curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.0/install.sh | bash

```
nvm install --lts
```

```
nvm use --lts
```