CSS Fundamentals

Peter J. Jones
✉ pjones@devalot.com
@devalot
http://devalot.com

DEVALOT

What's In Store

Day 1	Day 2
HTML & CSS Refresher	Positioning
Advanced Selectors	Floating
Images and Fonts	Responsive Design
Transforms	Flexible Grids
Transitions	Flexible Box
Animations	Tools and Frameworks

Web Browser

Text Editors or IDEs

Web Sites

HTML Refresher

What is HTML?

- Hyper Text Markup Language
- HTML is very error tolerant (browsers are very forgiving)
- That said, you should strive to write good HTML
- Structure of the UI and the content of the **view data**
- Parsed as a tree of nodes (elements)
- HTML5
  - Rich feature set
  - Semantic (focus on content and not style)
  - Cross-device compatibility
  - Easier!

Anatomy of an HTML Element

- Also known as: nodes, elements, and tags:

```
<element key="value" key2="value2">
  Text content of element
</element>
```

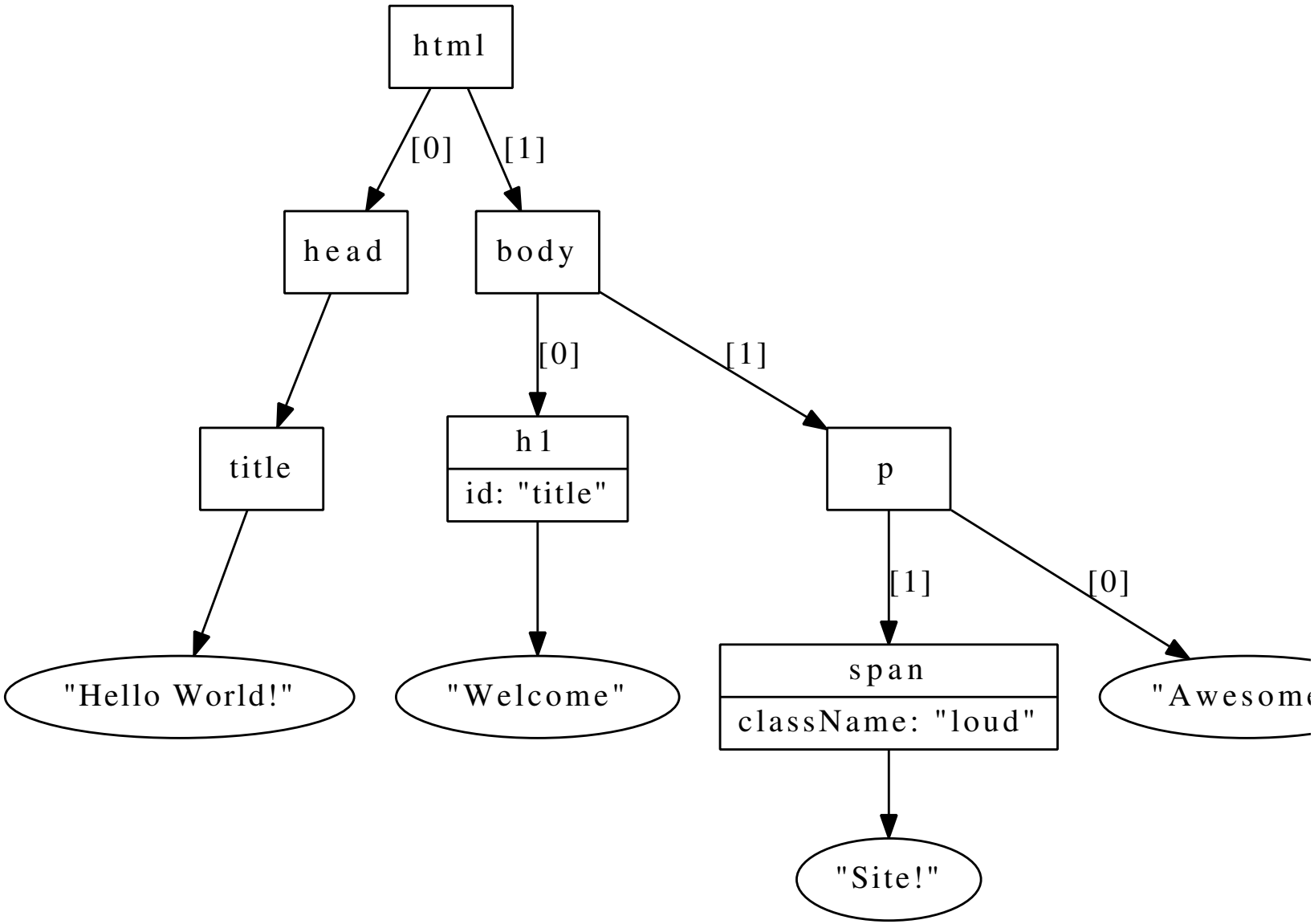
HTML Represented as Plain Text

```
<html>
  <head>
    <title>Hello World!</title>
  </head>

  <body>
    <h1 id="title">Welcome</h1>

    <p>
      Awesome <span class="loud">Site!</span>
    </p>
  </body>
</html>
```

HTML Parsed into a Tree Structure



CSS Refresher

What is CSS?

- Cascading Style Sheets
- Rule-based language for describing the look and formatting
- Separates presentation from content
- Can be a separate file or inline in the HTML
- Prefer using a separate file

What Does CSS Look Like?

```
p {
  background-color: white;
  color: blue;
  padding: 5px;
}

.spoiler {
  display: none;
}

p.spoiler {
  display: block;
  font-weight: bold;
}
```

Anatomy of a CSS Declaration

- Selectors choose which elements you want to style. A selector is followed by a body where styling properties are set:

```
selector {
  property-x: value;
  property-y: val1 val2;
}
```

- For example:

```
h1 {
  color: #444;
  border: 1px solid #000;
}
```

The Various Kinds of Selectors

- Using the element's type (name):
  - HTML: `<h1>Hello</h1>`
  - CSS: `h1 {...}`
- Using the ID attribute:
  - HTML: `<div id="header"></div>`
  - CSS: `#header {...}`
- Using the class attribute:
  - HTML: `<div class="main"></div>`
  - CSS: `.main {...}`
- Using location or relationships:
  - HTML: `<ul><li><p>One</p></li><li>Two</li></ul>`
  - CSS: `ul li p {...}`

Which Selectors Do You Already Know?

Selector Quiz

What do these selectors match, and what's the difference between them?

```
p span  { /* ... */ }
```

```
p, span { /* ... */ }
```

Selector Quiz

What does this selector match?

```
li.active.tracked span { /* ... */ }
```

Selector Quiz

What does this selector match?

```
ul li:first-child { /* ... */ }
```

Selector Quiz

What does this selector match?

```
li:nth-child(3n+1):not(:only-child) { /* ... */ }
```

ID and Class Selectors

The ID Selector

- HTML:

```
<section id="advertisement">
  <p>Buy now!</p>
</div>
```

- CSS:

```
#advertisement {
  font-weight: bold;
}
```

The Class Selector

- HTML:

```
<div class="info admin report">
  <p>Admin Report (blue).</p>
</div>

<div class="info report">
  <p>Normal Report (green).</p>
</div>
```

- CSS:

```
div.info.report {
  color: green;
}

div.info.admin.report {
  color: blue;
}
```

Siblings, Children, and Descendants

Descendant Selector

- HTML:

```
<article>
  <ul><li>...</li></ul>
  <section>
    <ul><li>...</li></ul>
  </section>
</article>
```

- CSS:

```
article ul { /* ... */ }
```

- Match all `<ul>` decedents of `<article>`

Child Selector

- HTML:

```
<article>
  <ul><li>...</li></ul>
  <section>
    <ul><li>...</li></ul>
  </section>
</article>
```

- CSS:

```
article > ul { /* ... */ }
```

- Match <ul> decedents of <article> that are direct children

Sibling Selector

- HTML:

```
<h2>Hello There!</h2>
<p>Paragraph 1.</p>
<p>Paragraph 2.</p>
```

- CSS:

```
h2 + p { /* ... */ }
```

- Match the <p> elements that immediately follow a <h2> (next sibling)

General Sibling Selector

- HTML:

```
<p>Hello!</p>
<ul><li>...</li></ul>
<ol><li>...</li></ol>
```

- CSS:

```
p ~ ol { /* ... */ }
```

- Match all <ol> siblings that come after a <p>

Exercise: Siblings, Children, and Descendants

1. Open (and edit) the following file in your text editor:
   src/www/css/selectors/part-01.css
2. Review (but don't edit) the following file:
   src/www/css/selectors/index.html
3. Follow the directions in the CSS file
4. Open the HTML file in your browser and confirm your changes

Pseudo Classes and Elements

Pseudo What?

- Advanced selectors that use the element's state or relative location
- Can also select non-elements (e.g., paragraph text)
- Begin with a colon (`:`) instead of a dot (`.`)
- (Pseudo elements now start with two colons (`::`))

Pseudo Class Example

```
input:focus {
  border: 3px solid blue;
}
```

Pseudo Element Example

```
/* First (visible) line: */
p::first-line {
  color: red;
}

/* First character: */
p::first-letter {
  font-size:   4em;
  font-weight: bold;
}
```

Partial List of Pseudo Classes and Elements

Classes	Elements
`:link`	`::first-line`
`:visited`	`::first-letter`
`:active`	`::after`
`:checked`	`::before`
`:focus`	`::selection`
`:hover`	
`:enabled`	
`:disabled`	
`:root`	

Exercise: Pseudo Classes and Elements

1. Open (and edit) the following file in your text editor:
   src/www/css/selectors/part-02.css
2. Review (but don't edit) the following file:
   src/www/css/selectors/index.html
3. Follow the directions in the CSS file
4. Open the HTML file in your browser and confirm your changes

Child Pseudo Selectors

Selecting the First or Last Child

- HTML:

```
<ul>
  <li>First</li>
  <li>Second</li>
  <li>Third</li>
  <li>Forth</li>
</ul>
```

- CSS:

```
li:first-child, li:last-child {
  background-color: #eee;
}

li:only-child {
  color: #f00;
}
```

Selecting First or Last Based on Type

- HTML:

```
<section class="products">
  <header><h2>Products</h2></header>
  <p>First</p>
  <p>Second</p>
  <p>Third</p>
</section>
```

- CSS:

```
.products p:first-of-type {
  border-top: 1px solid #ddd;
}

.products p:last-of-type {
  border-bottom: 1px solid #ddd;
}
```

Exercise: Child Pseudo Selectors

1. Open (and edit) the following file in your text editor:
   src/www/css/selectors/part-03.css
2. Review (but don't edit) the following file:
   src/www/css/selectors/index.html
3. Follow the directions in the CSS file
4. Open the HTML file in your browser and confirm your changes

Pseudo Classes that Take Values

Selecting Any Grouping of Children

```
:nth-child(value) { /* ... */ }
```

Example uses of `nth-child`:

- Select even or odd children
- Select every third child
- Select the first 5 children
- Select the last 8 children

Even or Odd Children

```
li:nth-child(even) { /* ... */ }
li:nth-child(odd)  { /* ... */ }
```

(The first child is odd.)

The Nth Child

Select the third (and only the third) child:

```
li:nth-child(3) { /* ... */ }
```

Every Nth Child

Select the third child and every third child after that:

```
li:nth-child(3n) { /* ... */ }
```

Every Nth Child Starting at X

Select every third child, starting at the first child:

```
li:nth-child(3n+1) { /* ... */ }
```

Selecting All Previous or Following Children

Select all children after (and including) the second child:

```
li:nth-child(n+2) { /* ... */ }
```

Select all child before (and including) the second child:

```
li:nth-child(-n+2) { /* ... */ }
```

Nth Child Variations

`:nth-last-child`: Starts from the bottom of the child list.
`:nth-of-type`: Filters the child list by a type selector.
`:nth-last-of-type`: `:nth-of-type` + `:nth-last-child`

The `not` Pseudo Class

Negating a Selector

```
ul:not(.products) {
  background-color: #eee;
}
```

Simple Selectors

The `:not` pseudo-class can only be used with *simple selectors*:

- Type selector
- Universal selector
- Attribute selector
- Class and pseudo-class selectors
- ID selector

Type selector example:

```
.products:not(ul) {
  background-color: #f00;
}
```

Exercise: Pseudo Classes that Take Values

1. Open (and edit) the following file in your text editor:
   src/www/css/selectors/part-04.css
2. Review (but don't edit) the following file:
   src/www/css/selectors/index.html
3. Follow the directions in the CSS file
4. Open the HTML file in your browser and confirm your changes

Attribute Selectors

Selecting Based on Arbitrary Attributes

Writing a selector for the `id` or `class` attributes is easy. What about the other attributes?

```
/* Attribute exists */
input[placeholder] {
  color: #eee;
}

/* Attribute has exact value */
input[type="number"] {
  border: none;
}

/* Attribute contains substring */
a[href*="salesforce.com"] {
  font-weight: bold;
}
```

Available Operators

Operator	Description	Example
`=`	Exact match	`[type="text"]`
`~=`	Contains word	`[class~="foo"]`
`|=`	Prefix before dash	`[lang|="en"]`
`^=`	Begins with	`[href^="http://"]`
`$=`	Ends with	`[href$=".pdf"]`
`*=`	Contains substring	`[href*="salesforce.com"]`

Exercise: Attribute Selectors

1. Open (and edit) the following file in your text editor:
   src/www/css/selectors/part-05.css
2. Review (but don't edit) the following file:
   src/www/css/selectors/index.html
3. Follow the directions in the CSS file
4. Open the HTML file in your browser and confirm your changes

Form Styling with CSS

Form Validation in the Browser

Validation attributes:

- `max`: Maximum number or date
- `maxlength`: Maximum number of characters
- `min`: Minimum number or date
- `minlength`: Minimum number of characters
- `pattern`: Regular expression `value` must match
- `required`: Input must have a value
- `title`: Describe the `pattern` conditions

CSS Pseudo Classes:

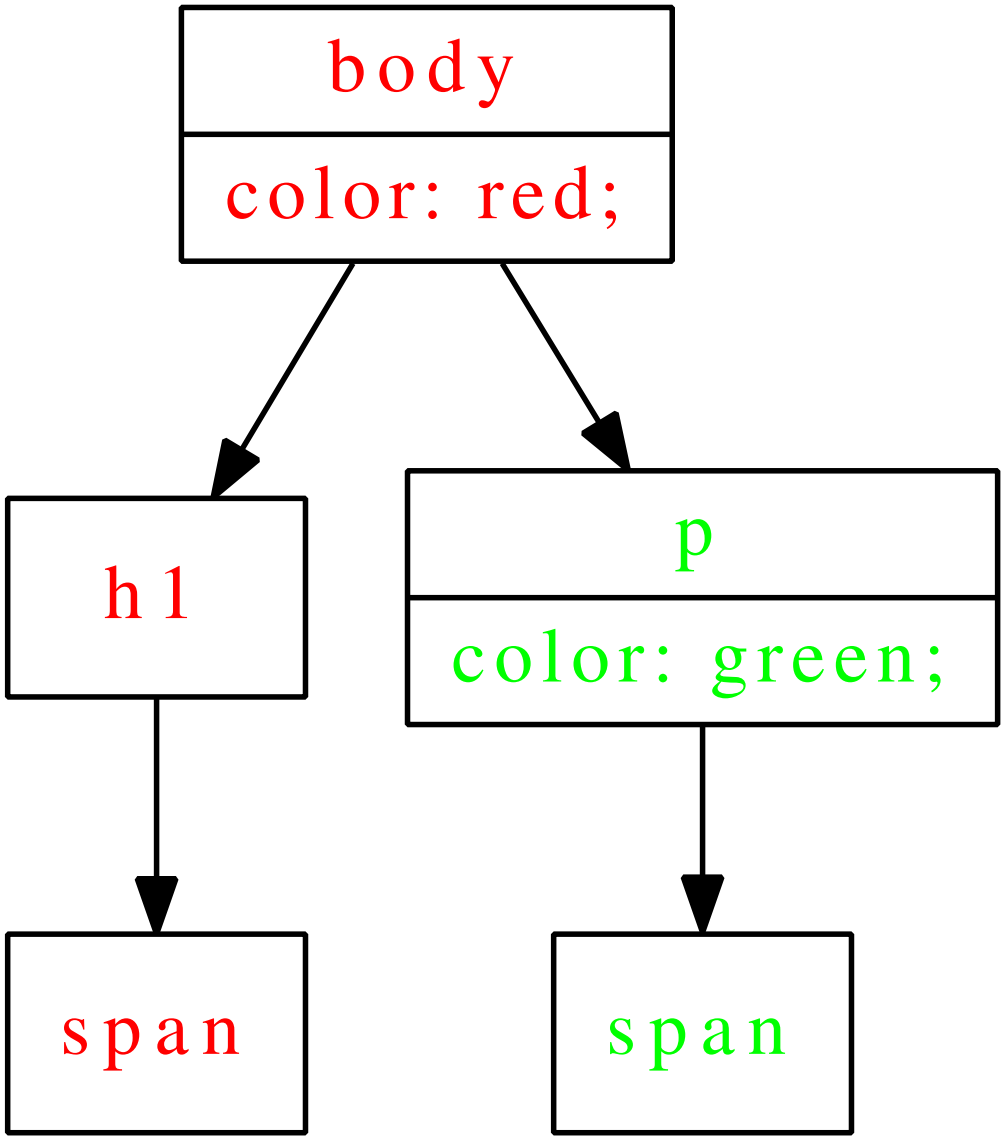
- `:valid`: Element's value is valid
- `:invalid`: Element's value is invalid
- `:optional`: No value is required
- `:required`: A value is required

Exercise: Form Styling

1. Open (and edit) the following files in your text editor:
   - `src/www/css/form/form.css`
   - `src/www/css/form/index.html`
2. Follow the directions in the CSS file
3. Open the HTML file in your browser and confirm your changes

Inheritance

Inheriting Styles from Ancestors



Inheritable Properties

An (incomplete) list of inheritable properties

- `line-height`
- `color`
- `text-align`
- `letter-spacing`
- `font-family`
- `font-style`
- `font-variant`
- `font-weight`
- `font-size`

Forcing Inheritance

You can inherit any value from a parent as long as it's set on the parent and you use the `inherit` value keyword:

```
/* Set the value on the parent:       */
p          { border: 1px solid blue;   }

/* Inherit from the parent:           */
p > span { border: inherit;           }

/* Or change the property's behavior */
*          { border: inherit;          }
```

The Cascade

Conflicting Properties

What happens when properties conflict?

- HTML:

```
<div id="main" class="fancy">
  What color will this text be?
</div>
```

- CSS:

```
#main {color: red;}

#main.fancy {color: blue;}

div.fancy {color: green;}
```

Specificity

Specificity Chart

Selector	Points	Examples
Universal selector	0	`*`
Type selectors	1	`p`, `a`, `h1`, etc.
Pseudo elements	1	`::before`, `::after`, etc.
Classes	10	`.sidebar`
Pseudo classes	10	`:nth-child`
Attribute selectors	10	`[type="number"]`
ID selectors	100	`#main`

- Inline styles add 1,000 points.
- Tie breaker: last defined style wins.
- Force highest specificity with `!important`.

Advanced Font Tricks

Specifying Fonts

```
html {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 16px;
}
```

Using Web Fonts

```
/* Create a new font-family */
@font-face {
  font-family: "My Font Name";
  src: url(/fonts/myfont.woff);
}

/* Then use it */
html {
  font-family: "My Font Name";
}
```

Web Font Services

Example using Google Fonts:

- HTML:

```
<link
 href="https://fonts.googleapis.com/css?family=Indie+Flowe
 rel="stylesheet">
```

- CSS:

```
html { font-family: "Indie Flower"; }
```

Using Images in CSS

Background Image Properties

`background-image`: The URL of the image.

`background-position`: Absolute or relative position of background.

`background-origin`: Controls where the background image is initially placed. That is, it's upper-left origin.

`background-size`: Constrain the size of an image, or scale the image up or down.

`background-repeat`: How to tile images smaller than the container.

`background-clip`: Which bounding box the background (image or color) will be clipped to.

`background-attachment`: Control image location when scrolling.

Exercise: Background Images

1. Open (and edit) the following file in your text editor: src/www/css/background-image/index.css
2. Review (but don't edit) the following file: src/www/css/background-image/index.html
3. Modify the CSS so that your browser shows the same page as the one on the instructor's screen
4. Open the HTML file in your browser and confirm your changes

Image Sprites

- Several images stored in a single file
- Size the parent element to the size of a single image
- Changing `background-position` will change which image is show
- Can be animated with CSS or JavaScript

Exercise: Icons

1. Open (and edit) the following file in your text editor: src/www/css/icons/index.css
2. Review (but don't edit) the following file: src/www/css/icons/index.html
3. Make each <LI> show only it's designated icon.
4. Open the HTML file in your browser and confirm your changes

Embedded Images

Images can be encoded using Base64 and directly embedded in a CSS file:

```
.logo {
  background-image:
    url(-DATA-GOES-HERE);
}
```

CSS Animation Basics

Major Animation Components

Transforms:	Primitive operations like rotation and scaling
Transitions:	Animate the change between two sets of styles
Animations:	Complex animations between any number of styles

Transforms

Requesting a Transform

```
.side-banner {
  transform: rotate(90deg);
}
```

Transform Operations

Rotation:	`rotate(90deg)`: Positive rotation is clockwise
Scaling:	`scale(2)`: Multiply current size by the given number
Translation:	`translate(10px, 10px)`: Move by given amount
Skewing:	`skew(15deg, 0)`: Slant lines by the given angle

Putting It All Together

- Transformed elements don't affect the flow of other elements (i.e. they leave a hole)
- The default origin for transformations is the center of the element and can be changed with:

```
.foo { transform-origin: left top; }
```

- Multiple transforms can be specified:

```
.foo { transform: rotate(45deg) scale(0.9); }
```

Transitions

Transition Ingredients

1. Two styles, the *beginning* style and the *ending* style
2. Typically the ending style uses a pseudo-class such as `:hover`
3. When triggered the browser will animate the transition
4. The `transition-property` and `transition-duration` properties are placed on the beginning style

Transition Example

```
.jumper {
  transition-property: all;
  transition-duration: 500ms;
}

.jumper:hover {
  transform: scale(2);
  transition-property: all;
  transition-duration: 250ms;
}

.jumper:active {
  transform: scale(0);
}
```

Transition Timing

- The `transition-duration` property controls how long the entire animation will last.
- The `transition-timing-function` property controls the rate at which the animation progresses and changes
- Built-in values include:
  - `ease` (default)
  - `ease-in`
  - `ease-out`
  - `ease-in-out`
  - `linear`
  - `cubic-bezier` (gives you total control)

Animations

Animations: Better Transitions

- Like transitions except you can have more than two styles
- Can be triggered like transitions or started on page load
- Easier to reuse with other elements
- Better timing control compared to transitions

Defining Animation Steps

Animations are given a name and series of steps (known as keyframes) using the `@keyframes` *at rule*:

```
@keyframes colorPlay {
  from { color: green;  }
  25%  { color: blue;   }
  75%  { color: purple; }
  to   { color: red;    }
}
```

Using an Animation

An animation can be added to any element in order to get it to start when the page loads:

```
.standout {
  animation-name: colorPlay;
  animation-duration: 5s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
}
```

You can also trigger an animation using a pseudo-class or via JavaScript.

Animation Properties

`animation-duration`: Total length of time the animation runs from start to finish (0%–100%).

`animation-timing-function`: Control rate of change (can also be used in keyframes to override the timing function for each stage of the animation).

`animation-delay`: Optional time to wait before starting the animation (default is `0s`).

`animation-iteration-count`: Number of times to run the animation, or `infinite` to continually repeat the animation.

`animation-direction`: Direction through the keyframes (the `alternate` value means to go forwards and backwards). Other values include `normal` and `reverse`.

`animation-fill-mode`: Which state to leave the element in (`forwards` means keep the element in the state it was in at the last keyframe).

Introduction to Page Layout

What is CSS Page Layout?

- HTML files specify a bunch of boxes with content
- The browser needs to arrange those boxes on the screen
- Arrangement is performed based on a set of layout rules
- The layout can be changed in CSS, per box

Layout Engines

We'll be looking at the following layout engines:

- Block (`display: block|inline`)
- Positioned (`position: absolute|relative|fixed`)
- Floating (`float: left|right`)
- Flexible Box (`display: flex`)

Block Layout

The Default: Block/Inline

- Elements are either block or inline
- Block elements stack on top of one another
- Inline elements flow inside a block element
- This is the default layout engine
- Good for articles, not so good for applications

Introduction to the Box Model

Open the following file in your web browser:

`www/box-model/index.html`

- Block elements have newlines before and after their content
- Inline elements flow in the content of a block element.

Positioned Layouts

Positioned Boxes

- Boxes can be pulled out of the normal flow of the HTML
- You can position them in specific locations
- Goal: achieve better compatibility with print designs

Absolutely Positioned Boxes

- Boxes are completely removed from flow of the page
- They can be positioned using any corner of the box
- Position is relative to nearest positioned ancestor
- Good for placing elements relative to a parent
- Good for images that flow/stack over other element

Relatively Positioned Boxes

- Boxes are moved from current location, leaving a “hole”
- They can be positioned using any corner of the box
- Position is relative to the boxes original location
- Mostly used to set an anchor point for absolutely positioned children

Fixed Position Boxes

- Locked to a specific screen location
- Scrolling the page doesn't change box location
- Boxes are completely removed from the page flow
- Position is relative to browser window
- Good for fixed navigation bar or page banner

Stacking Issues

- Positioning leads to boxes stack on top of other boxes
- You can control the stacking order with the `z-index` property
- The larger the value the higher a box is in stack
- Negative `z-index` values can be used to force a box to be underneath all other boxes

Floating Layouts

Floated Boxes

- Boxes can be floated so they are side-by-side with their siblings
- Sibling boxes will wrap around the floated box
- Boxes can be floated to the left or the right

Using the Floating Layout

Float boxes with the `float` property:

```
.sidebar {
  float: left; /* left, right, or none  */
  width: 25%;  /* remember to set width */
  margin-right: 1em; /* Push the main content away */
}


footer {
  clear: both; /* Stop the floating */
}
```

Problem: Float Drop

- Boxes are dropping below the floated box instead of side-by-side
- Set a width for all of the floated boxes
- Keep the box model in mind (border, margin, padding, etc.)
- You can also make the browser include the entire box in the width:

```
* {
  box-sizing: border-box;
}
```

Problem: Floating Siblings

- Floated boxes can escape their parent and continue to float other boxes (when the floated box is the biggest child)
- Make the parent enclose and clear the float:

```
.container::after {
  content: " ";
  display: table;
  clear:   both;
}
```

Responsive Design

So Many Browser Sizes, One HTML File

- Fixed design: Treating the web like paper
- Liquid design: Better but more complicated
- Responsive: Adapt to each browser

Mobile Browsers and Zooming

- Mobile browsers automatically zoom out to show all content
- The first step to making a site responsive is to disable this
- Use the following `meta` tag in the `head` of your HTML:

```
<meta name="viewport" content="width=device-width">
```

- With that, browsers will respect width requests without zooming

Relative Measurements

- Avoid using absolute units such as `px`, `pt`, `cm`, `in`, `mm`, etc.
- Relative to current font size: `em`
- Relative to parent element size: `%`
- Percentages + Media Queries = Responsive Web Design

Introduction to Media Queries

- Media Queries are part of CSS
- They are like `if` statements in your CSS
- Example:

```
/* If the browser window is at least 400px wide... */
@media (min-width: 400px) {
  .sidebar {
    float: left;
    width: 25%;
  }
}
```

Compound Media Queries

Media queries can be combined with `and`:

```
@media (min-width: 400px) and (orientation: portrait) {
  /* ... */
}
```

Media Queries and Breakpoints

Set media query breakpoints—divisions of screen width that change the CSS:

```
@media (max-width: 480px) {
  /* Small screens */
}

@media (min-width: 481px) and (max-width: 768px) {
  /* Medium screens */
}

@media (min-width: 769px) {
  /* Larger screens */
}

/* Etc. */
```

Mobile First, or Desktop First?

There are two ways to approach responsive web design:

1. Design for small screens and use media queries to adapt the design for larger screens
2. Start with a design for large screens and use media queries to scale the design down to smaller screens

Fluid Images

- Automatically scale images to match the container width:

```
img { max-width: 100%; }
```

- For this to work, don't use `width` or `height` attributes on an `img` tag:

```
<img src="logo.jpg" alt="Logo">
```

Flexible Grids

Designing with a Grid

A powerful design technique from the print world involves using a grid to divide the page into rows and columns. This also works well for the web.

- Slice the page into a series of rows
- Each row is then split into columns
- The number of columns varies from row to row

Flexible Grid Example

- The first row contains two columns:
  1. Company logo (50%)
  2. Site navigation (50%)
- The next row contains three columns:
  1. Left sidebar (25%)
  2. Main content (50%)
  3. Right sidebar (25%)
- The final row contains a single column:
  1. The footer (100%)

Grid Systems

- Straight forward to make responsive:
  - Small screens are limited to one column
  - Bigger screens can have more columns
- Automatically add space between columns
- Usually divide the screen into twelve units
- Columns can occupy between one and twelve units
- Class names map to unit numbers:

```
<div class="three columns">
```

Flexible Boxes

A Layout Engine for the Modern Web

- Easy to use with visually pleasing defaults
- Similar to a grid system, but easier to use
- No weird CSS tricks or class names to learn
- Universally supported (IE $>=$ 11)

Flexible Boxes: The Basics

- Mark a container element as a flexible box:

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

- All children then become *flex items*
- Flex items can be laid out in rows or columns
- Wide range of alignment, sizing, and wrapping options

Flex Item Layout

- Items side-by-side, left to right (default):

```
.container {
  display: flex;
  flex-direction: row;  /* or row-reverse */
}
```

- Items stacked top to bottom:

```
.container {
  display: flex;
  flex-direction: column; /* or column-reverse */
}
```

Flex Direction Orientation

Since flex can layout items in a row or a column it uses generic terms to refer to its axes:

- Main axis vs. cross axis
  - For row: main is horizontal, cross is vertical
  - For column: main is vertical, cross is horizontal
- Main start and end, vs. cross start and cross end
  - For row: main start is on the left
  - For row-reverse: main start is on the right

Flex Item Wrapping

- Items must all be on the same line (row):

```
.container {
  display: flex;
  flex-wrap: nowrap; /* This is the default */
}
```

- Items are allowed to wrap onto the next line:

```
.container {
  display: flex;
  flex-wrap: wrap; /* or wrap-reverse */
}
```

Flex Item Sizing

The `flex-grow`, `flex-shrink`, and `flex-basis` properties control the width of flex items relative to their siblings.

- Make all flex items the same width:

```
.container {
  display: flex;

  /* flex-grow flex-shrink flex-basis */
  flex: 1 1 250px;
}
```

- Make the second item take up twice as much space as the others:

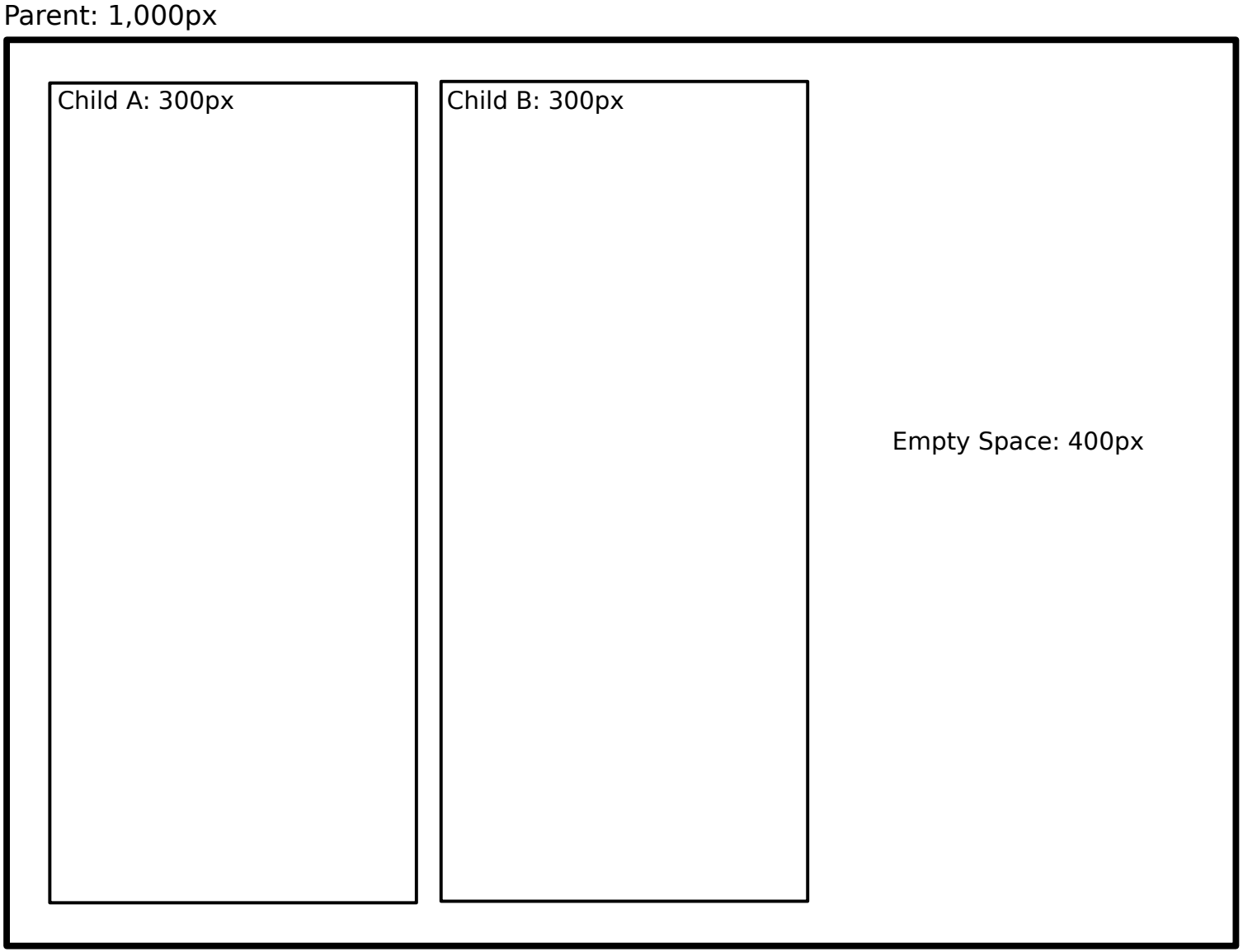
```
.container                { display: flex; }
.container > *            {    flex: 1;    }
.container :nth-child(2) {    flex: 2;    }
```

A Word About `flex-basis`

This property can be a bit tricky to understand. It's not your fault, it's complicated!

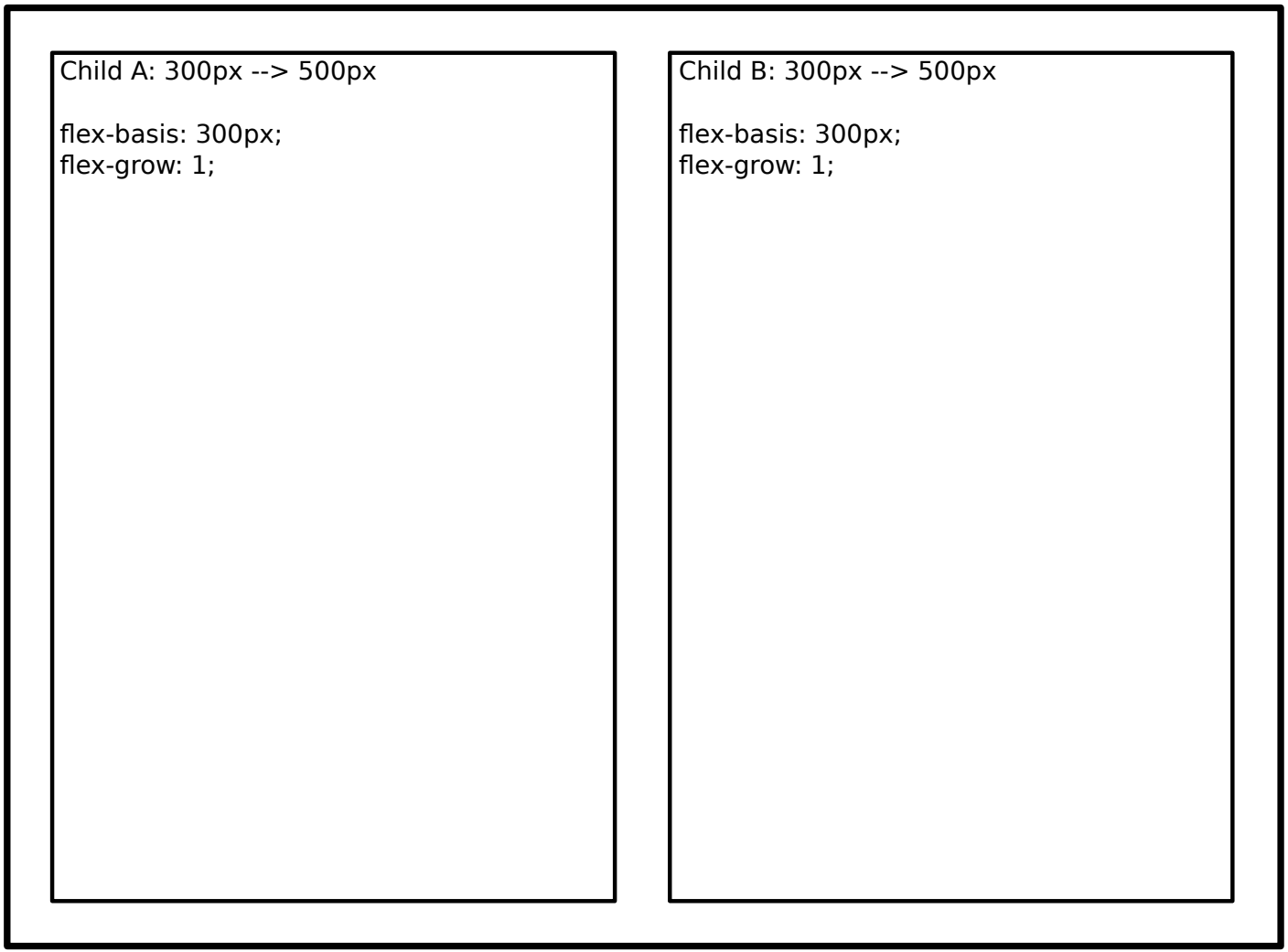
- Definition: The *initial* size of a flex item.
- The default value (today): `auto`
- Most common value: An absolute or relative measurement
- Future values: `min-content`, `max-content`, `stretch-fit`.

Flex Grow: Extra Space



Flex Grow: Uniform Growth

Parent: 1,000px

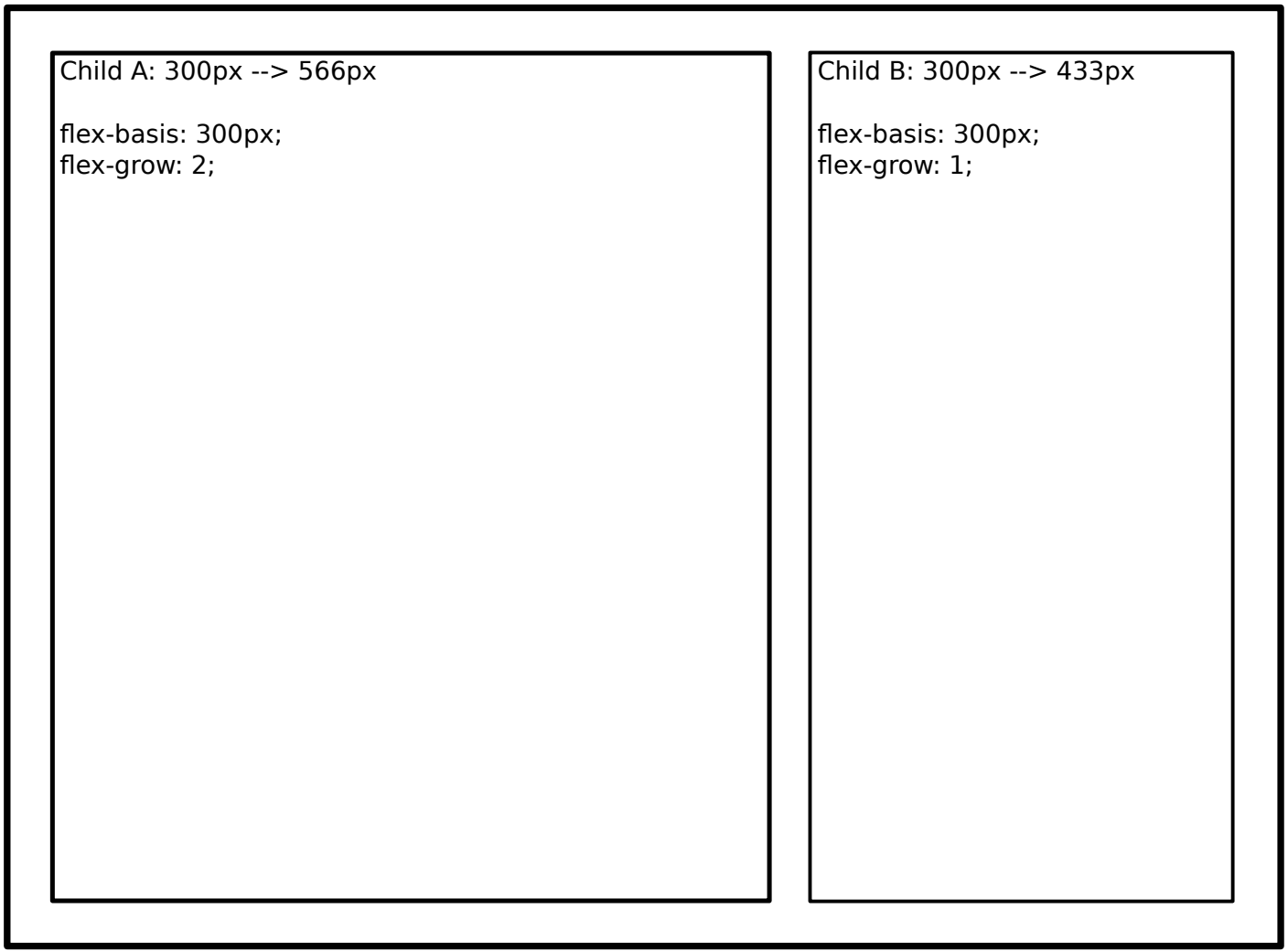


grow = Empty Space * (Child Flex Grow / Total Flex Grow)

Child Grow = 200px = 400px * (1 / 2)

Flex Grow: Nonuniform Growth

Parent: 1,000px



grow = Empty Space * (Child Flex Grow / Total Flex Grow)

Child A Grow = 266px = 400px * (2 / 3); Child B Grow = 133px = 400px * (1 / 3)

Flex Item Ordering

Items inside a flex container can be displayed in a different order than they appear in the HTML source code. This is done with the `order` property:

```
.container          { display: flex; }
.sidebar.primary    {   order: 1;    }
.main               {   order: 2;    }
.sidebar.secondary  {   order: 3;    }
```

Flex Alignment

`justify-content`: How space is distributed around and between items on the main axis. Requires that all flex items have a `flex-grow` of zero.
Useful values; `space-between`, `space-around`, `space-evenly`, `flex-start` (the default), and `flex-end`.

`align-items`: How space is distributed around and between items on the cross axis. Used when flex items have different cross axis sizes.
Useful values: `stretch` (the default), `center`, `flex-start` `flex-end`.

`align-content`: How space is distributed around and between multiple lines on the main axis created by wrapping.
Useful values: `stretch` (the default), and all values form `justify-content`.

Flex Container and Item Properties

Container	Item
`flex-flow`	`order`
`flex-direction`	`align-self`
`flex-wrap`	`flex`
`justify-content`	`flex-grow`
`align-items`	`flex-shrink`
`align-content`	`flex-basis`

Preprocessors

What Does a Preprocessor Do?

- Preprocessors add extra features to CSS or provide a totally different styling language for you to use
- They read your styling file and produce a standard CSS file
- A few will even validate your CSS against the standard
- Can automatically add vendor prefixes as necessary, etc.

Introduction to Sass

Syntactically Awesome Style Sheets (Sass) is an extension language to CSS providing several features:

- Variables (an extremely useful feature)
- Predefined functions for math, color blending, string manipulation, etc.
- Rule nesting (place one selector inside another)
- Property nesting (avoid repeating property prefixes)

Sass Variables

Typically, variables are set at the top of the file, or in a separate file:

```
$main-background-color: #eee;
$main-foreground-color: #888;
```

Then used throughout the rest of the file:

```
body {
  background-color: $main-background-color;
  color: $main-foreground-color;
}
```

CSS without Preprocessors: Variables

```
:root {
  /* Set some CSS variables: */
  --primary-color:   #0000ff;
  --secondary-color: #ff0000;
}

.banner {
  /* Expand a variable via `var': */
  background-color: var(--primary-color);
}
```

Popular Preprocessors

- Autoprefixer (2013)
- Sass (2006)
- Less (2009)
- Myth (2013)

Frameworks

Bootstrap

- Provides a flexible grid system
- Built in response design
- Includes styling for common components
- Lots of websites use Bootstrap, and therefore look very similar

Popular Frameworks

- Bootstrap
- Foundation
- Compass
- Bourbon
- Susy

Official Documentation

Books

Cheat Sheets

Training Videos from Pluralsight