Front End Developer Bootcamp

2014

What we'll cover

- Basics of HTML5 & CSS/3,
- JavaScript, jQuery
- Introduction to Fireworks
- Introduction to Knockout JS
- Building using Grunt
- Chrome Dev Tools

And what we won't cover…

- Large frameworks like Ember or Angular
- Dependency management libraries like Require.js or Common.js
- CSS preprocessors such as LESS or SASS/SCSS
- Cross browser testing
- JavaScript testing frameworks
- Phoenix

The goal this week

- Build the front end to a single page application
- But more on that later…

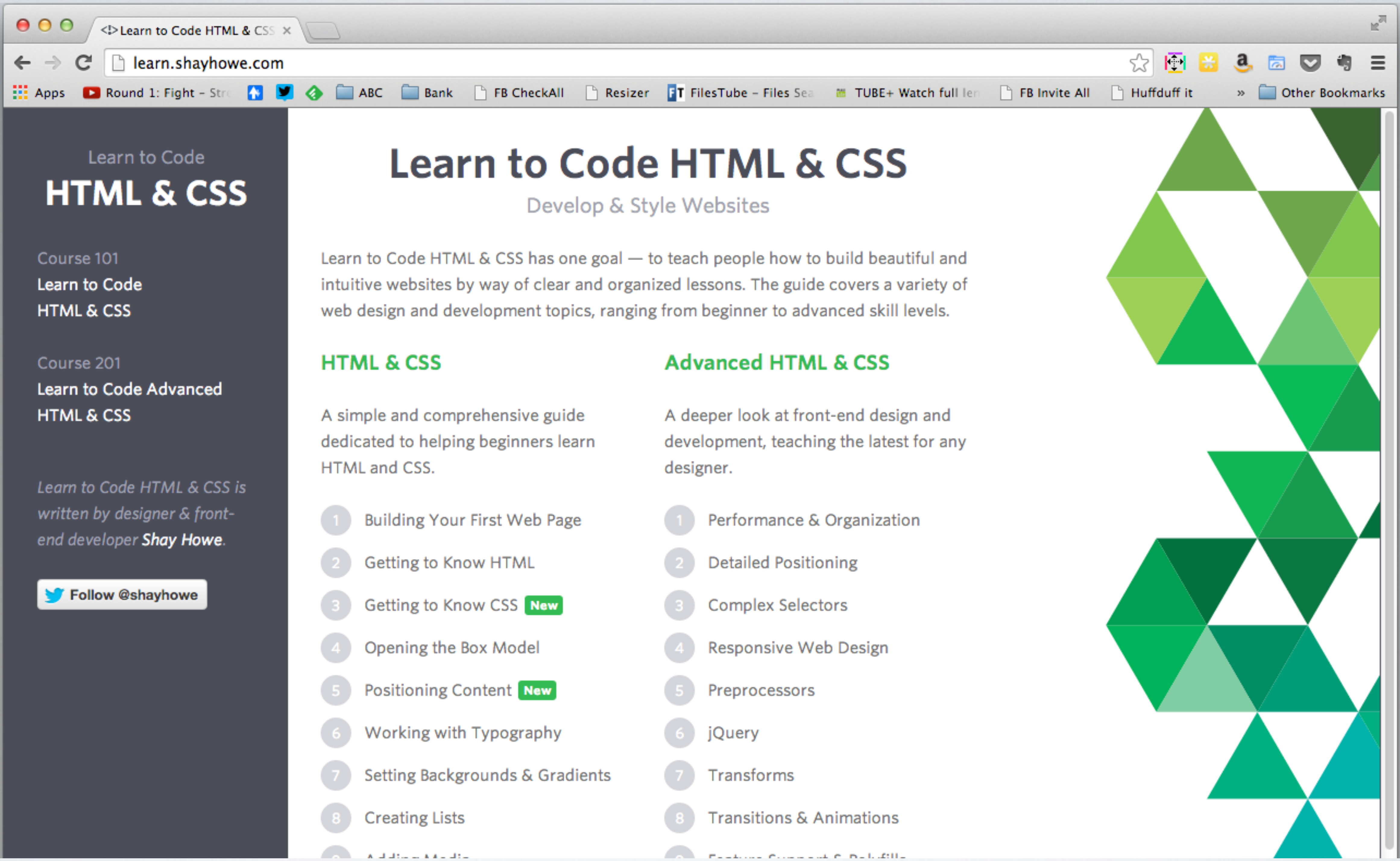
Tools you'll need

- **JetBrains WebStorm (trial)**
  www.jetbrains.com/webstorm/download/

- **Node JS**
  nodejs.org/download/

- **Adobe Fireworks (trial)**
  https://creative.adobe.com/products/fireworks

- **Grunt CLI**
  npm install -g grunt-cli



Learn to Code
HTML & CSS

Course 101
Learn to Code
HTML & CSS

Course 201
Learn to Code Advanced
HTML & CSS

*Learn to Code HTML & CSS is written by designer & front-end developer **Shay Howe**.*

Follow @shayhowe

Learn to Code HTML & CSS

Develop & Style Websites

Learn to Code HTML & CSS has one goal — to teach people how to build beautiful and intuitive websites by way of clear and organized lessons. The guide covers a variety of web design and development topics, ranging from beginner to advanced skill levels.

HTML & CSS

A simple and comprehensive guide dedicated to helping beginners learn HTML and CSS.

1. Building Your First Web Page
2. Getting to Know HTML
3. Getting to Know CSS New
4. Opening the Box Model
5. Positioning Content New
6. Working with Typography
7. Setting Backgrounds & Gradients
8. Creating Lists

Advanced HTML & CSS

A deeper look at front-end design and development, teaching the latest for any designer.

1. Performance & Organization
2. Detailed Positioning
3. Complex Selectors
4. Responsive Web Design
5. Preprocessors
6. jQuery
7. Transforms
8. Transitions & Animations

learn.shayhowe.com

(Props to this dude)

I’ve never done this before!

Approximate timeline

- Monday: HTML & Basic CSS
- Tuesday: CSS layout
- Wednesday: jQuery
- Thursday: Knockout
- Friday: Fit & finish and whatevers left

"It's just a bunch of divs and spans, right?"

–Mike Wallace

…not *the* Mike Wallace,
but the Advisory Board developer Mike Wallace

Role of FEDs in ABC

- Part of the Member Experience (MX) Group
- MX group includes:
  - UX designers & researchers
  - Visual designers
  - Front end developers

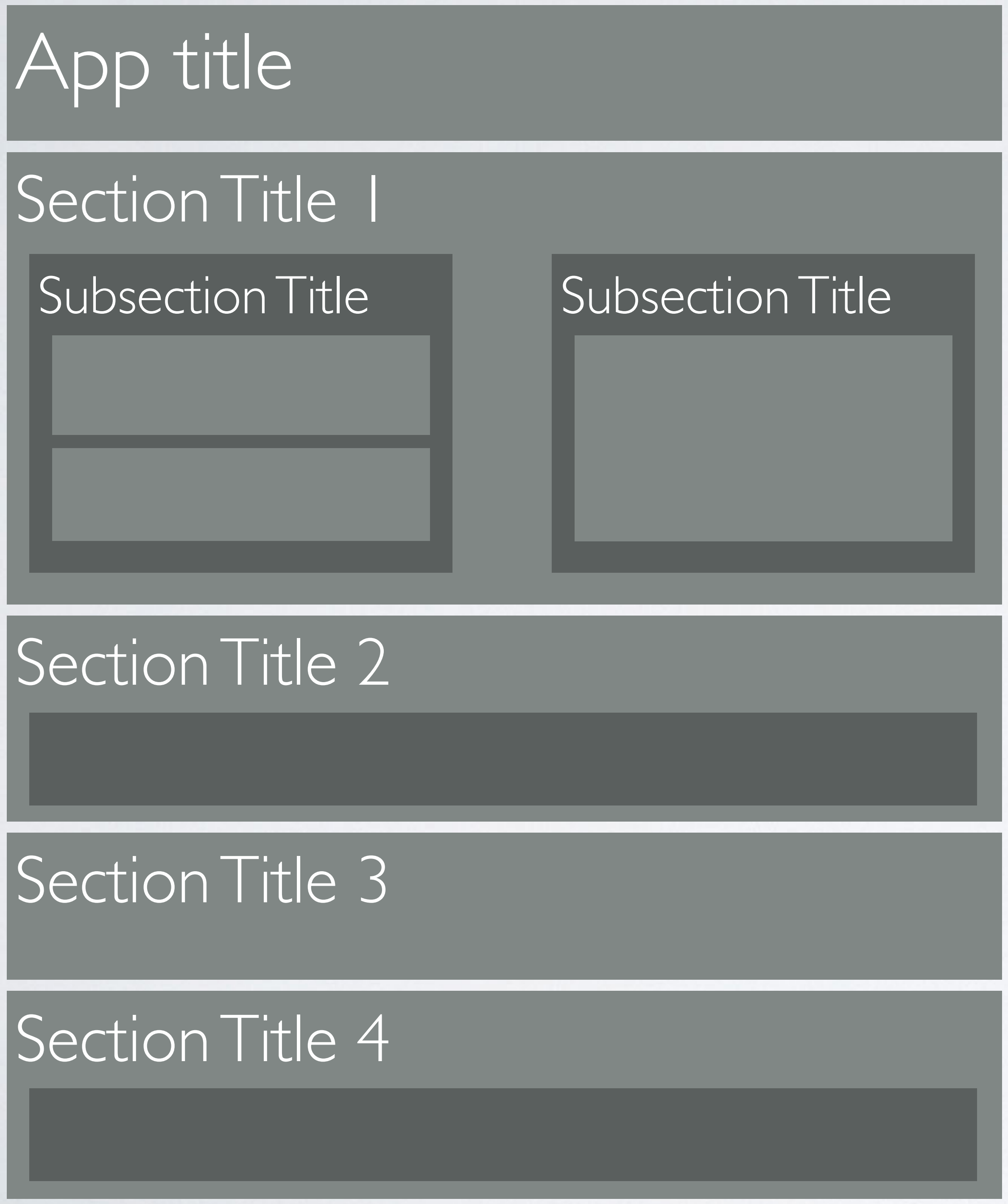
What we’re gonna build

The goods

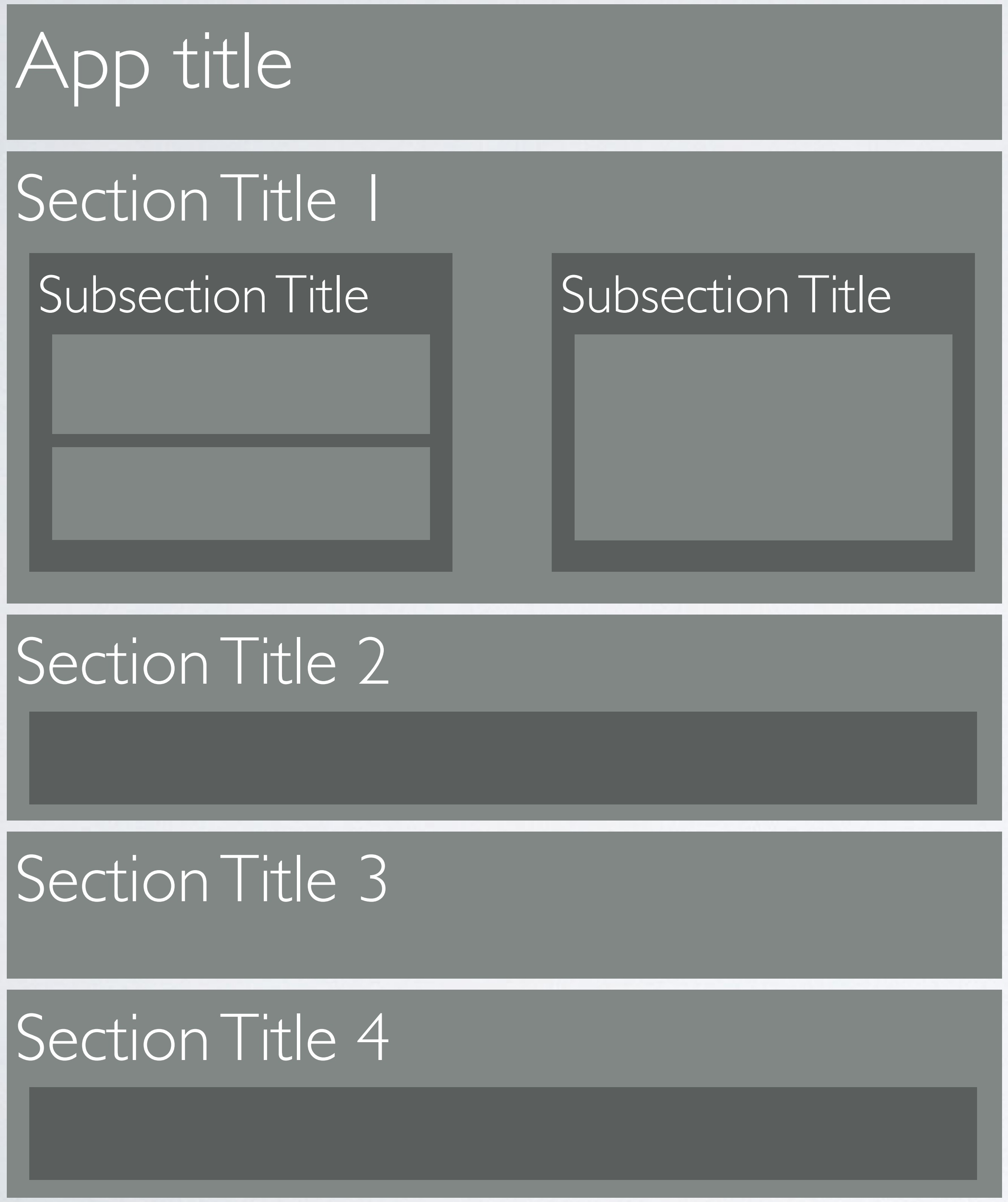
Stash this gem away to refer back to…

https://github.com/ericmasiello/fed-training

Design > Outline > HTML

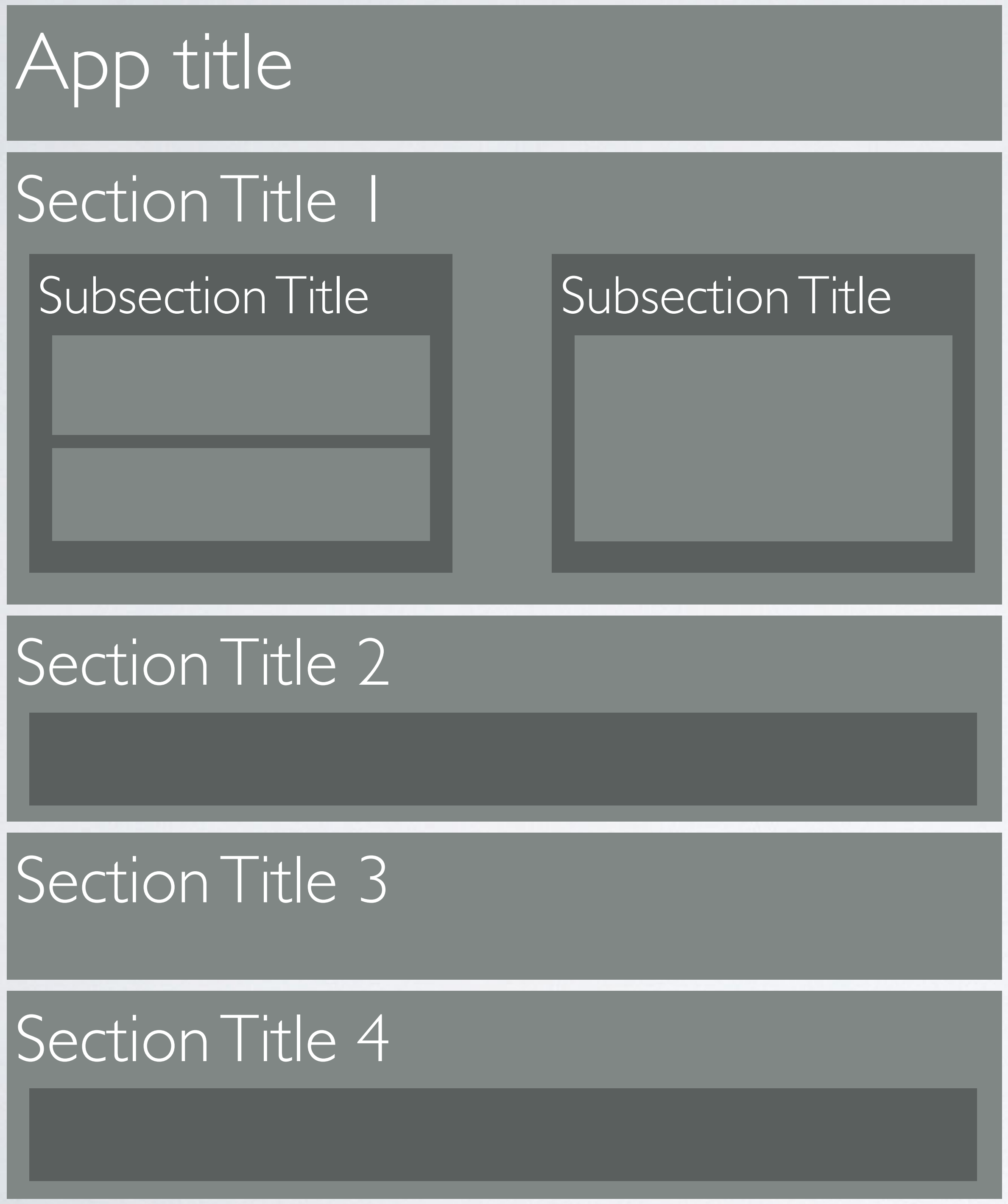


Design > Outline > HTML



A. Primary Title
    A. Section Title 1
        A. Subsection Title
            A. Content
            B. Content
        B. Subsection Title 2
            A. Content
    B. Section Title 2
        A. Content
    C. Section Title 3
    D. Section Title 4
        A. Content

Design > Outline > HTML



```
<header><h1>App Title</h1></header>
<section>
     <h1>Section Title 1</h1>
     <div>
          <h2>Subsection Title</h2>
          <p>Content</p>
     </div>
</section>
<section>
     <h1>Section Title 2</h1>
     <form></form>
</section>
<section>
     <h1>Section Title 3</h1>
</section>
<section>
     <h1>Section Title 4</h1>
</section>
```

Design > Outline > HTML

- A. Primary Title
    - A. Section Title 1
        - A. Subsection Title
            - A. Content
            - B. Content
        - B. Subsection Title 2
            - A. Content
    - B. Section Title 2
        - A. Content
    - C. Section Title 3
    - D. Section Title 4
        - A. Content

```
<header><h1>App Title</h1></header>
<section>
     <h1>Section Title 1</h1>
     <div>
          <h2>Subsection Title</h2>
          <p>Content</p>
     </div>
</section>
<section>
     <h1>Section Title 2</h1>
     <form></form>
</section>
<section>
     <h1>Section Title 3</h1>
</section>
<section>
     <h1>Section Title 4</h1>
</section>
```

20 mins

Exercise Zero

- Build an outline for our project

HTML & CSS

HTML: More than divs and spans

- Provide structure and organization to your site/application
- Provides meaning not only to the browser but those using assistive technology (e.g. screen readers)

"Semantics within HTML is the practice of giving content on the page meaning and structure by using the proper element. Semantic code describes the value of content on a page, regardless of the style or appearance of that content. There are several benefits to using semantic elements, including enabling computers, screen readers, search engines, and other devices to adequately read and understand the content on a web page. Additionally, semantic HTML is easier to manage and work with, as it shows clearly what each piece of content is about"

– http://learn.shayhowe.com/html-css/getting-to-know-html/

CSS

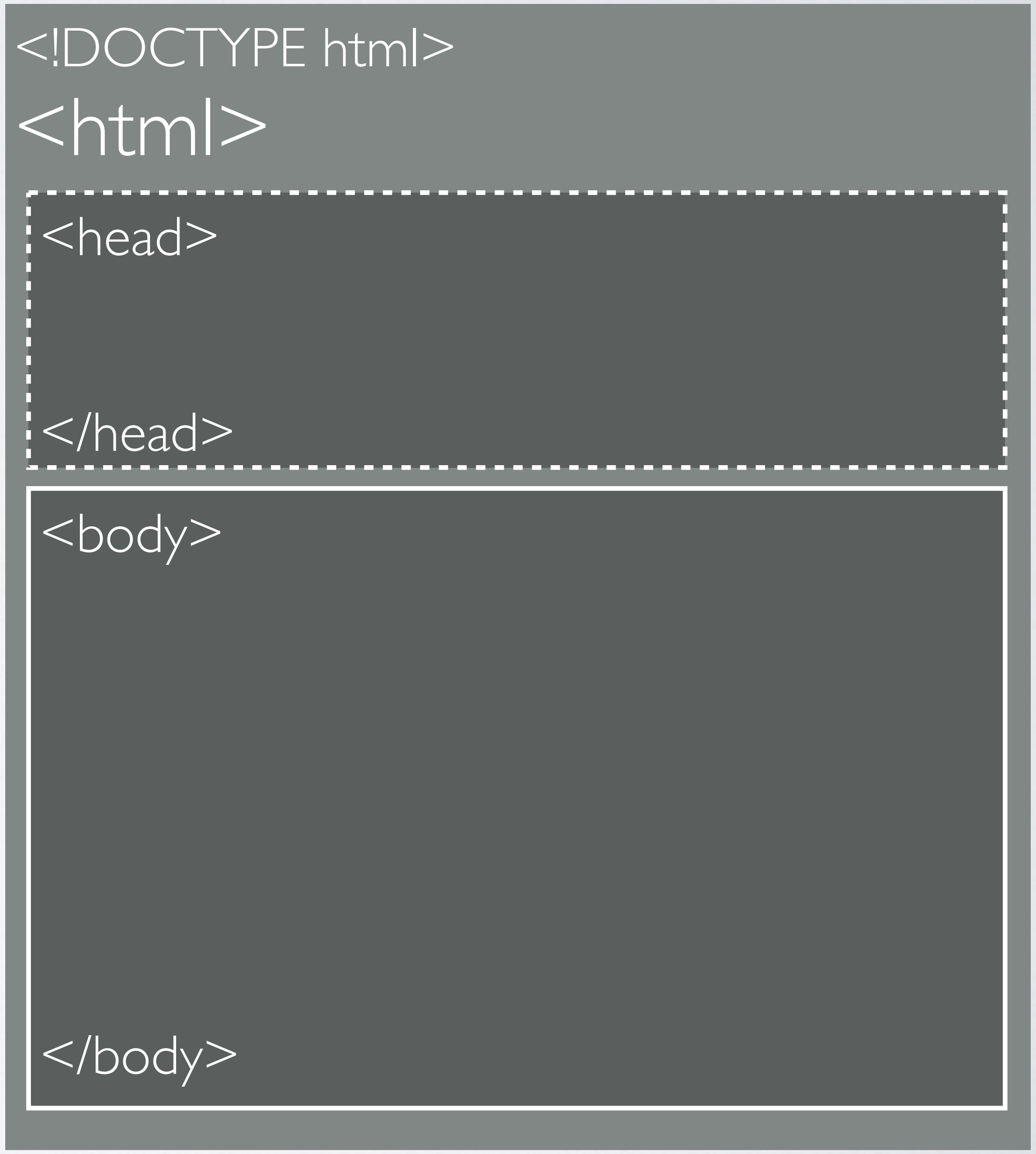
- Presentation language to style the appearance of the content
  - Typography
  - Color and design elements
  - Layout

Validating your code

- HTML: http://validator.w3.org/
- CSS: http://jigsaw.w3.org/css-validator/

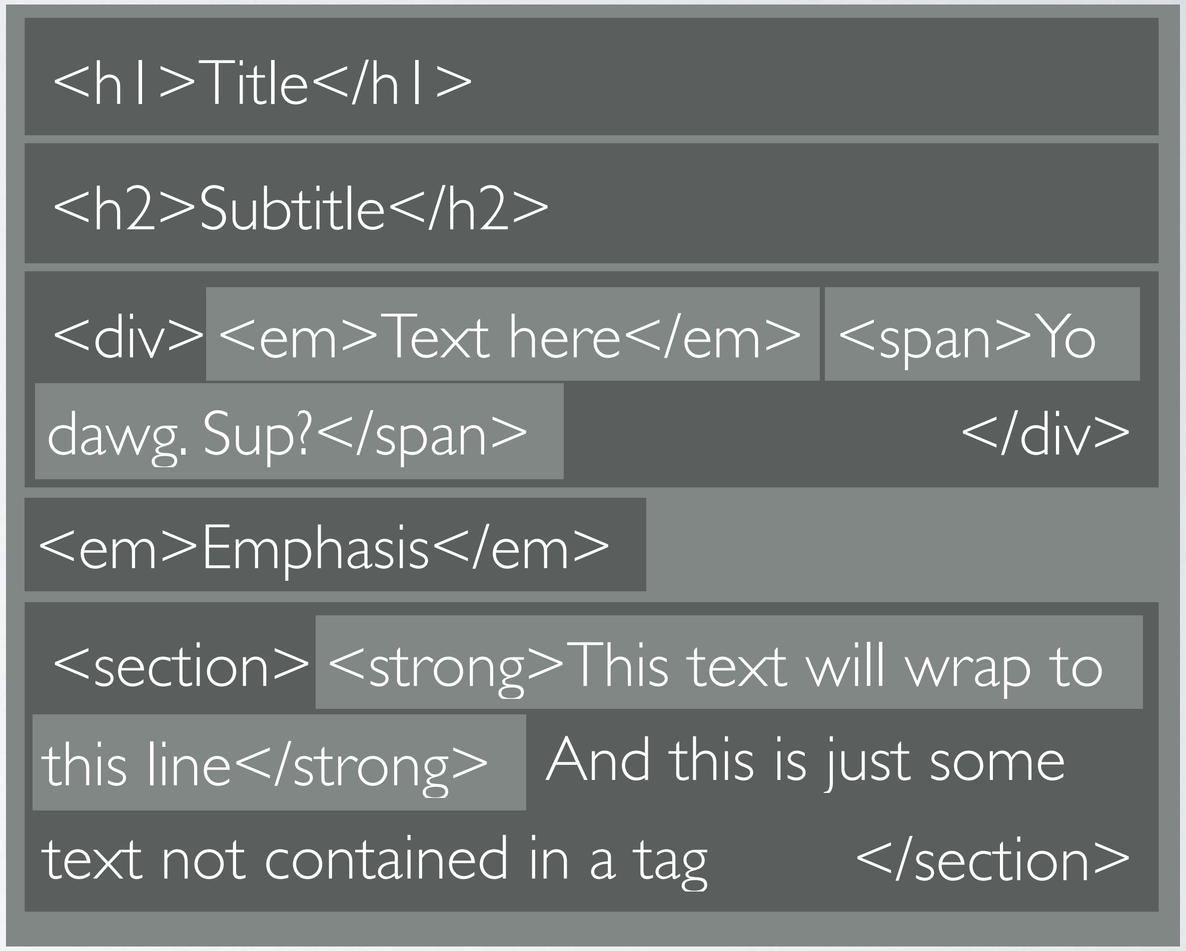
Document structure

- DOCTYPE: tells browser version of HTML
- <HTML> container for entire document
- <HEAD>
    - invisible element
    - meta data
    - title of document
    - links to external files (*.css, *.js)
- <BODY>
    - visible part of the document



Block vs. inline

- Block level elements
  - e.g. h1, h2, h3, p, div, etc
  - take up 100% width by default
  - start a new line
  - typically used for larger pieces of content.
  - can contain all inline level elements as many block elements *(cannot contain a <p> inside of a <p>)
- Inline level elements
  - e.g. em, strong, span
  - fall into the "normal flow" of a document and do not begin a new line.
  - typically cannot contain block level elements *

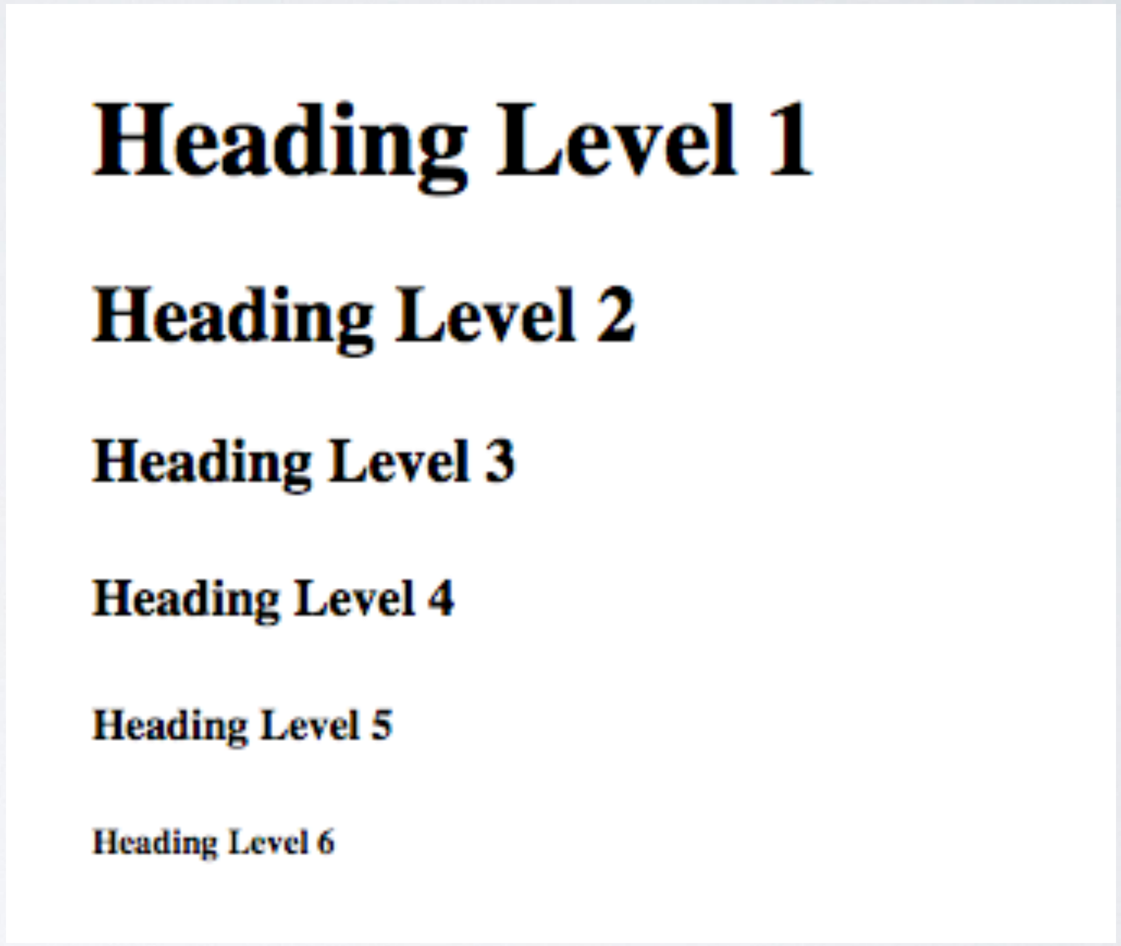


<h1>Title</h1>
<h2>Subtitle</h2>
<div> <em>Text here</em> <span>Yo dawg. Sup?</span> </div>
<em>Emphasis</em>
<section> <strong>This text will wrap to this line</strong> And this is just some text not contained in a tag </section>

See CSS box model

Headings

- Help set hierarchy of your document
- 6 levels
- Should flow in order
  - h2 then h3 - **cool**
  - h2 then h2 - **love it**
  - h3 then h6 - **no!**
- Should only have a single <h1> inside a structural element


Heading Level 1

Heading Level 2

Heading Level 3

Heading Level 4

Heading Level 5

Heading Level 6

Lists

- Ordered <ol>
- Unordered <ul>
- Definition <dl>
- Styling lists slide

```html
<!—— Has order ——>
<ol>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
</ol>
<!—— No specific order ——>
<ul>
    <li><a href=#>Variance Finder</a></li>
    <li><a href=#>Assignment Rules</a></li>
    <li><a href=#>My account</a></li>
</ul>
<!—— Definition or Key value pair ——>
<dl>
    <dt>FED</dt>
    <dd>Front end developer</dd>
</dl>
```

Tables

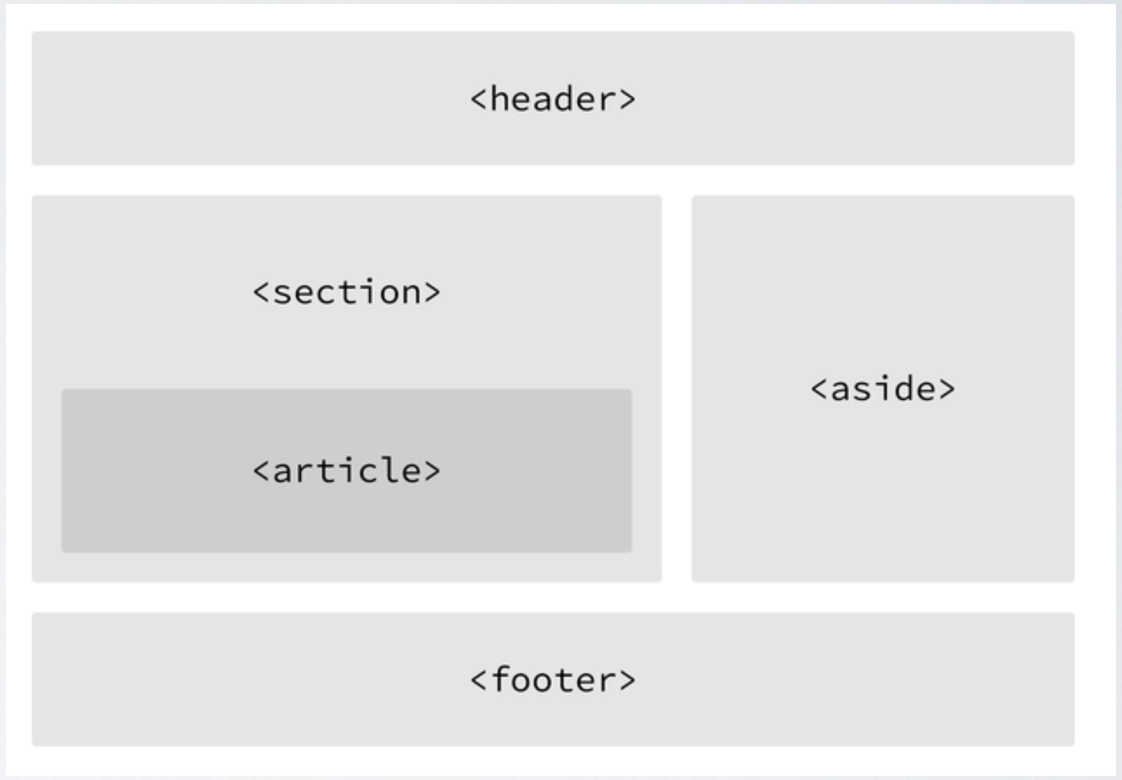
- <table>
- <col>
- <thead>
- <tfoot>
- <tbody>
- <tr>
- <th>, <td>

http://codepen.io/ericmasiello/pen/dsruz?editors=110

```
<table>
  <col>
  <col>
  <thead>
    <tr>
      <th scope=“col”>Header 1</th>
      <th scope=“col”>Header 2</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td colspan=“2”>I’m at the end</th>
    </tr>
  </tfoot>
  <tbody>
    <tr><td>Val</td> <td>Val</td></tr>
    <tr><td>Val</td> <td>Val</td></tr>
    <tr><td>Val</td> <td>Val</td></tr>
  </tbody>
</table>
```

HTML5 structural elements

- <header>
  - container for heading section of page or other structural elements
- <footer>
  - Close out the page, a <section>, <article>
  - Should contain info relevant to parent container
- <nav>
  - container for primary nav
- <section>
  - thematic grouping of content
  - often contains its own <header> or <footer>



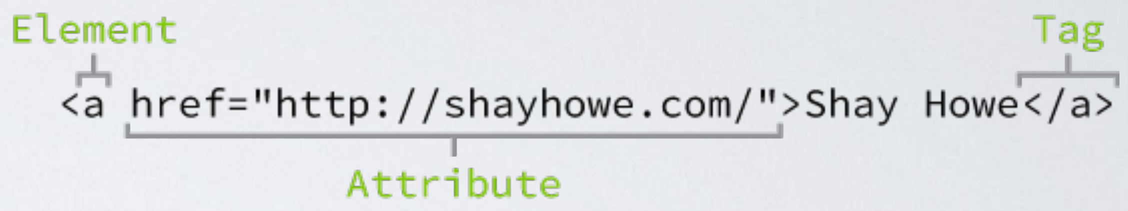
Non-semantic elements

- Useful when the content being grouped…
  - is solely for styling purposes
  - and doesn’t provide value to the outline of a document
- <div></div>
  - block level
- <span></span>
  - inline level

```
<div class=“bordered”>
I’m just here to add borders around this
content.
<h3>Header</h3>
<p>Guess what?
<span class=“success”>We won!</span>
</p>
</div>
```

HTML attributes

- Usually constructed with an attribute and value
  - <a **href="http://www.google.com"** >Link</a>
  - <img **src="/some/image.jpg"** />
  - <input **type="text"** />
- Sometimes values can be omitted
  - <input type="button" **disabled** />
  - <option value="123" **selected**>Eric</option>
- Data Attributes: (Invent your own attributes)
  - <div **data-whatever="hi there"**></div>

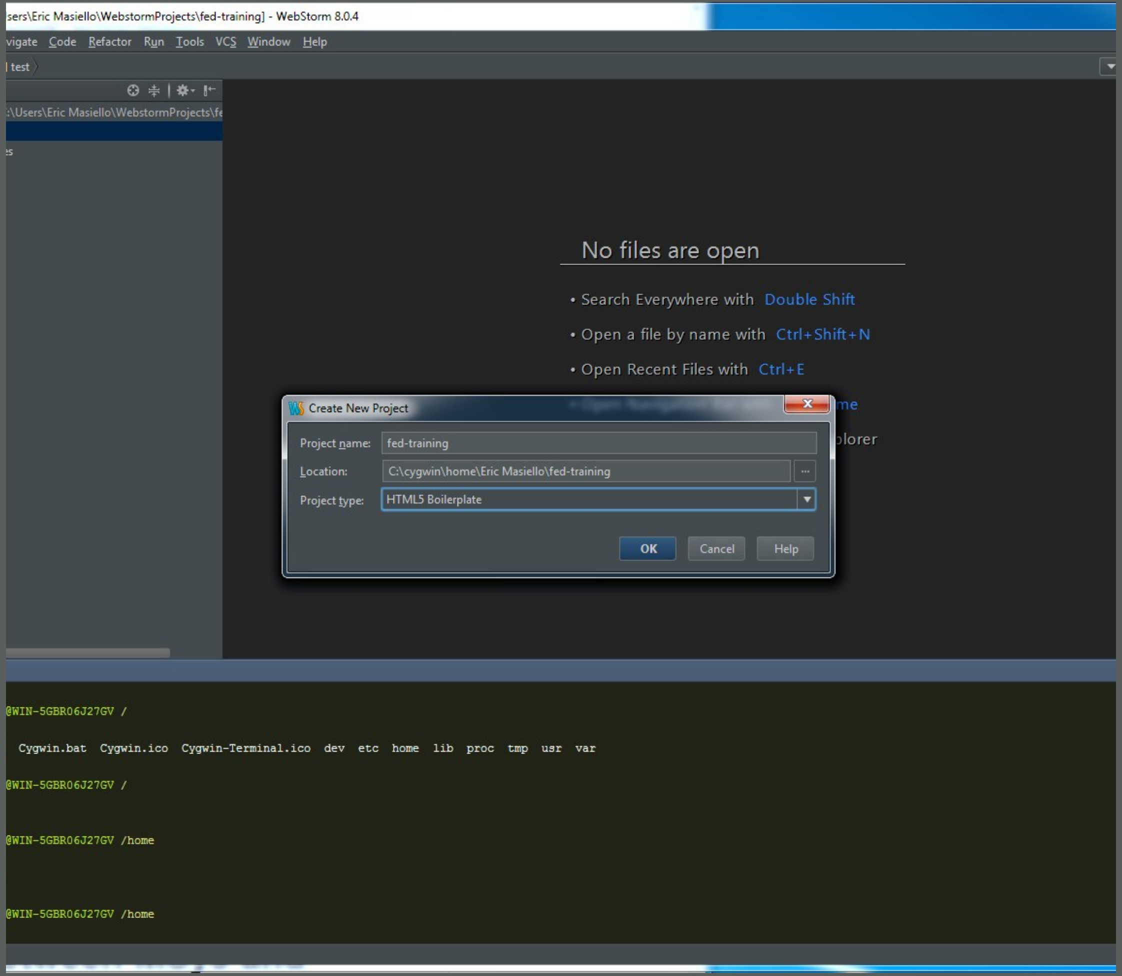


```
<a href="http://shayhowe.com/">Shay Howe</a>
```

40 mins

Exercise 1 & 2: project setup

- Inside WebStorm, create a new Project
- Select "HTML5 Boilerplate" as the project type
- **As a class:** add the following structural & container elements to your page: **header**, **section**, **nav**, **div**, **table**, **form**
- **On your own**: fill in the rest


sers\Eric Masiello\WebstormProjects\fed-training] - WebStorm 8.0.4

vigate Code Refactor Run Tools VCS Window Help
test
:\Users\Eric Masiello\WebstormProjects\fe

No files are open

- Search Everywhere with Double Shift
- Open a file by name with Ctrl+Shift+N
- Open Recent Files with Ctrl+E

Create New Project

Project name: fed-training
Location: C:\cygwin\home\Eric Masiello\fed-training
Project type: HTML5 Boilerplate

```
@WIN-5GBR06J27GV /

 Cygwin.bat  Cygwin.ico  Cygwin-Terminal.ico  dev  etc  home  lib  proc  tmp  usr  var

@WIN-5GBR06J27GV /


@WIN-5GBR06J27GV /home



@WIN-5GBR06J27GV /home
```

CSS

Types of selectors

- #id (avoid these)
- .class
- tag
- :state
- [attribute]
- :positional(selector)
- ::pseudo-elements

Demo

```
/* ids e.g. <div id="main-content"></div> */
#main-content { }

/* classes e.g. <div class="warning"></div> */
.warning { }

/* any HTML tag */
body, p, input {}

/* state */
a:visited {}, button:disabled {}

/* attribute */
[type="text"], [data-modal="open"]

/* positional */
p:first-child, p:last-child { }
tr:nth-child(2n) { }
tbody:nth-of-type(2) { }

/* pseudo elements, good for any tag that doesn't
self close */
q::before, q::after { content: '"' }
```

Specificity, Combinators, & Best Practices

- So what rules win?
  - Same level of specificity, last rule wins
  - Type selectors < Class Selectors < ID Selectors
- Combinators
  - Last element in a combo selector is the target selector
  - e.g. `table` **`td`** `{}` *or* `.myClassName` **`.thingy`** `{}` *or* `.myClassName` **`p`** `{}`
  - Combo selectors increases the specificity for the target selector
  - Other common combos : `> + ~`
  - **Demo**
- Best Practices
  - Keep selector specificity low
  - Combine lots of classes for highly portable/reusable styles (OOCSS)
  - **Demo**

Color properties

- hex:
  - color: #fff;
  - color: #efefef;
- named value:
  - background-color: red;
  - border-color: green;
- rgb/rgba:
  - color: rgb(0, 0, 0);
  - color: rgba(255, 255, 255, .7);

Typography properties

- font-family
- font-size
- font-style
- font-variant
- font-weight
  - normal (400), bold (700), any value between 100-900
- line-height

```
body {
  font-family: "Helvetica Neue", Helvetica,
               Arial, sans-serif;
  font-size: 12px;
  font-style: italic;
  font-variant: small-caps;
  font-weight: 600;
  line-height: 1.5em; /* 12 * 1.5 = 18px */
}
/* Shorthand */
body {
  font: italic small-caps 600 12px/1.5em
  "Helvetica Neue", Helvetica, Arial,
  sans-serif;
}
```

Typography properties continued

- text-align
- text-decoration
- text-indent
- text-shadow
- text-transform
- white-space
- vertical-align
- text-overflow
    - Demo

```
th {
  text-align: left;
  text-transform: uppercase
}
p {
  text-indent: 10px;
}
a:href {
  text-decoration: none;
}
a:hover {
  text-decoration: underline;
}
button {
  text-shadow: 3px 6px 2px
              rgba(0, 0, 0, .3);
}
```

CSS resets

- All browsers come with their internal, default styles
- May be different between browsers (IE vs. Chrome vs. Firefox)
- Resets allow you to reset/normalize these values across browsers
  - Eric Meyer's reset
    http://meyerweb.com/eric/tools/css/reset/
  - Nicolas Gallagher's normalize.css
    http://necolas.github.io/normalize.css/

Referencing CSS

- <link> inside <head>
- @import inside .css file

index.html

```
<html>
<head>
  <link rel="stylesheet" href="main.css">
</head>
<body></body>
</html>
```

main.css

```
@import url(“reset.css”)

html {
}

body {
}
```

A guided tour through Fireworks

30 mins

Exercise 3: color & type

1. Replace normalize.css with Eric Meyers' **reset.css**
2. Remove contents of main.css and **import your reset.css**
3. Make sure main.css is referenced in **index.html**. (Test page to see how it looks)
4. Go through Fireworks document and **take an inventory** of all the color values, font sizes, & line heights
5. Update your **main.css** with baseline **color** & **typography** styles

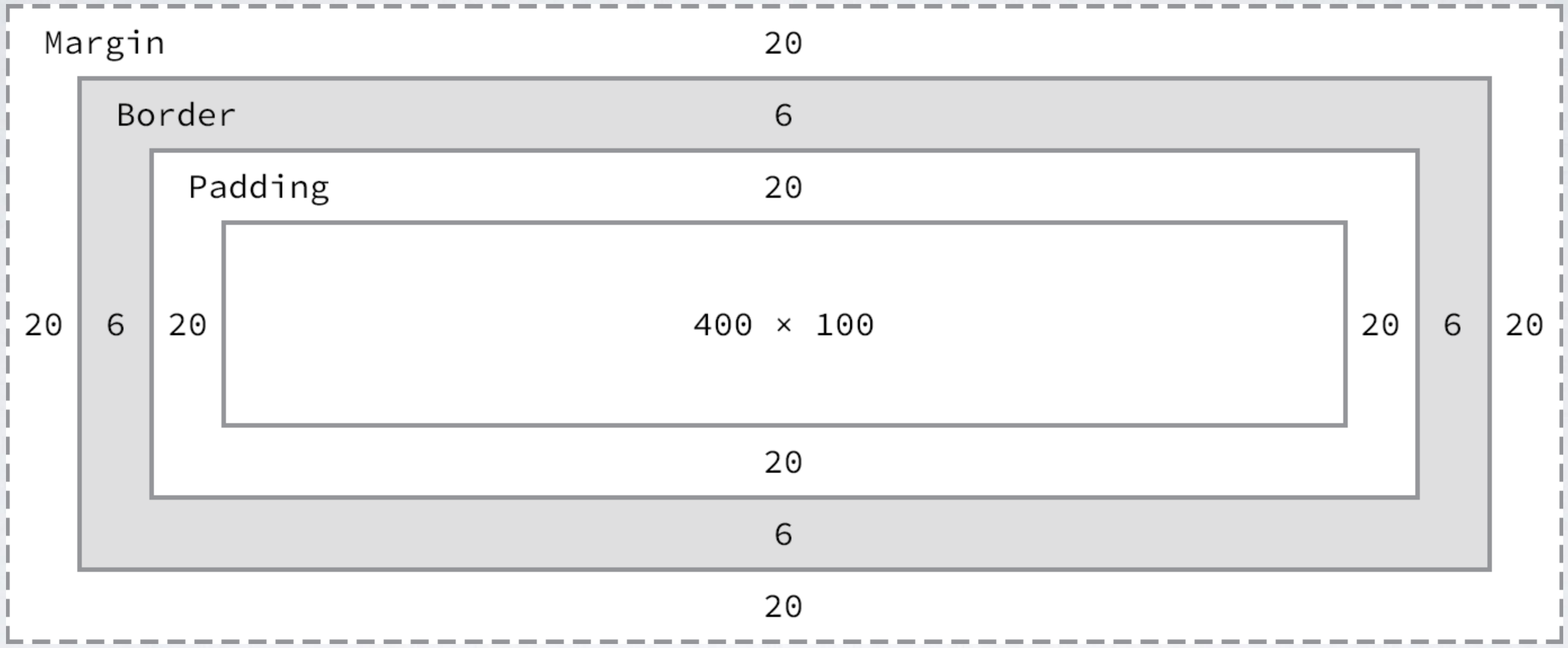
   Note and correct the following *(I'll help with these)*:

   - Certain form elements don't inherit your font styles - so force them to
   - Header font sizes aren't quite right: create classes to override (.h1, .h2, etc.)

CSS box model

- All tags/elements have a default display value
- Can be overwritten via css **display** property
- Common display values
  - block (see block vs inline)
  - inline
  - inline-block
  - none

Box model explained



```
div {
  /* default => box-sizing: content-box; */
  border: 6px solid #949599;
  height: 100px;
  margin: 20px;
  padding: 20px;
  width: 400px;
}
```

Box model applied

- width: [size value]
- height: [size value]
- box-sizing: [content-box | border-box]
- padding-[top | bottom | left | right]: [size value]
- margin-[top | bottom | left | right]: [size value]
- border-[top | bottom | left | right]-width: [size value]
- View Demo

Border properties

- Longhand:
  - border-style: solid | dotted | dashed
  - border-color: any color value
  - border-width: any width value
- Shorthand:
  - border-[top | bottom | left | right]-[style | color | width]
  - border: [width style color];

Shorthand

- One value applies to all sides: top, right, bottom, & left
  - padding: 10px
- Two values applies: top/bottom & left/right
  - margin: 5px 0
- Three values applies: top, left/right, & bottom
  - border-width: 5px 10px 8px
- Four values goes clockwise starting at the top then right, bottom & left
  - margin: 3px 4px 7px 2px

Pro tip

```
*, *::before, *::after {
  box-sizing: border-box;
}
```

Styling lists

- Review HTML slide
- list-style-type
- list-style-position
- View demo

```
ol {
  list-style-position: inside;
}

.undecorated {
  list-style-type: none;
}
```

40 mins

Exercise 4a & 4b: form styling

1. (Part04a) As a class:
    1. Build the CSS for our buttons including hover, active, and disabled state
    2. Add border-box wildcard rule
2. (Part04b) On your own or with a classmate:
    1. Hide the #manager div
       Feel free to use style="display: none" on the tag since this is temporary
    2. Apply styles to form section including
        1. Form elements except radio buttons, checkboxes, and select elements
        2. Apply list styles for whatever is appropriate:
        3. Don't forget hover and focus states to your form elements

Background color

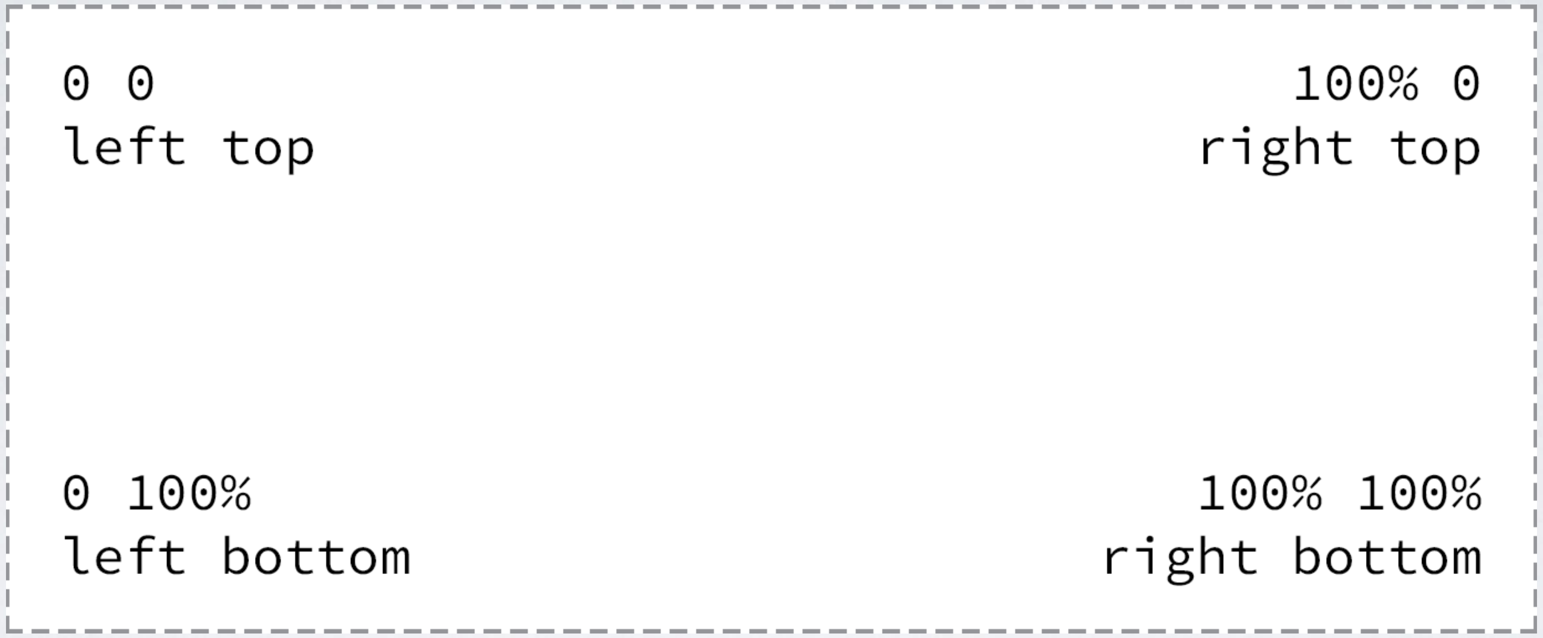
- Just like other color properties
- hex, keyword, rgb, rgba, etc
- *not all browsers support rgba so you can provide a fallback

```
.my-elm {

  /* fallback for older browsers */
  background-color: #333;

 /* browsers that understand rgba will
    override with this rule */
  background-color: rgba(0,0,0,.5);
}
```

Background image basics



```
.bg {
  background-image: url('/path/to/image.ext');
  background-repeat: no-repeat; /* or can be repeat, repeat-x, repeat-y */
  background-position: 0 0; /* same as left top */
  background-color: blue; /* color sits below the image */
}
.bg-shorthand {
  background: blue url('/path/to/image.ext') 0 0 no-repeat;
}
```

Background gradients

- Act like an image
- Use background-image to set them
- Types:
  - linear gradients
  - radial gradients

Linear gradients

- Use linear-gradient() in place of url()
- Default direction is top to bottom
- Specify direction with: to + [direction] or with degree value
- Colorzilla: online tool http://www.colorzilla.com/gradient-editor/
- Demo gradients

```
.gradient-examples {


  background-image: linear-gradient(to
right, #fff, #000);


  background-image: linear-gradient(to right
bottom, red, blue);


  background-image: linear-gradient(315deg,
rgba(255,255,255,.5), rgba(255,255,255,1));
}
```

Multiple backgrounds

- Demo

```
div {
  background:
  url("foreground.png") 0 0 no-repeat,
  url("middleground.png") 0 0 no-repeat,
  url("background.png") 0 0 no-repeat;
}
```

CSS position & layout

Floats

- Take elements out of the normal *flow* of a page
- Other elements on page will flow around the floated element
- Elements will float all the way to the edge of the parent
- Basic float demo

```
.fl {

  float: left;
}

.fr {

  float: right;
}

.fn {

  float: none;
}
```

Floats for layout

- Multiple items floated next to one another can be used to create columned layouts
- Floated elements by default constrain to the width of their content - set the width property to override that
- Floated elements take on box-model characteristics of display:block

1. Floats without widths demo
2. Floats with widths demo

Floats for layout are kinda a hack

- Originally designed for floating images within content – not for laying out columns
- Floats need to be *cleared*
  - Demo without clear and then demo with clear
- Or floats need to be *contained ("clearfix")*
  - Grid layout w/o clearfix and then demo with clearfix

Floats are kinda weird…
but also super useful

Inline block for layout

- display: inline-block;
- Inline block takes on box model characteristics of display block
- But still sorta acts like an inline element
- Demo fail
- Demo "no white space hack"
- Demo "comment hack"

Creating reusable layouts (grids)

- Create predefined sizes
- e.g. 25%, 50%, 75%, etc
- Create CSS class based system
- Create a gird system that aligns with the visual design system (.png)

```css
.grid-wrapper:before,
.grid-wrapper:after {
  content: "";
  display: table;
}
.grid-wrapper:after {
  clear: both;
}
.grid-wrapper {
  clear: both;
  *zoom: 1; /* For IE 6/7 only */
}

.grid {
  float: left;
}

.one-half {
  width: 50%;
}

.one-quarter {
  width: 25%;
}
```

Positioning

- position property establishes how an element flows in page
  - values: static, relative, absolute, fixed
  - everything defaults to static
- Use in conjunction with box offset properties
  - top, bottom, left, right

```
.abs {
  position: absolute;
  top: 10px;
  left: -10px; /* pulls to the left 10 */
  right: 0;
  bottom: 0;
}

.rel {
  position: relative;
  bottom: 10px; /* pushes up 10px */
  right: 5px; /* pushes right 10px; */
}

.static {
  position: static;
  left: 10px; /* no affect with static */
}
```

Relative position

- Allows elements to still appear in normal flow
- However still lets you modify their position with top, bottom, left, & right CSS properties
- Leaves a space where the element would have appeared in the normal flow
- Relative position demo

Absolute position

- Element will **not** appear in the normal flow
- Original space and position is **not** preserved
- Moved in relation to their closest parent *relative* or *absolute* positioned element
- If no parent element is defined as *relative* or *absolute*, position becomes relative to the <body> element
- Use z-index property to override stacking order
- Absolute demo
- Read more:
  - http://learn.shayhowe.com/html-css/positioning-content/
  - http://learn.shayhowe.com/advanced-html-css/detailed-css-positioning/

Questions?

60 mins

Exercises 5(a–d): some layout

1. (Part05a) As a class: using background and position we'll create webkit (Chrome/Safari) specific drop downs using hack

2. On your own or with a classmate (25 mins each)

    1. (Part05b) Style the main navigation tabs of the application

    2. (Part05c) Style the global nav (menu link + hamburger icon)
    Also add style for error icon add them next to a few fields

    3. (Part05d) Create a 12 column grid system. Use it to position the "Security Requirements" list
    Tip: Notice how when you shrink down your screen, the password rules overlaps the password fields. (Use max-width & width:100% to solve)

15 mins

Exercise 6a: refactor

1. Break down CSS into individual modules (files)
2. Group by 3 types:
    1. core
    2. components
    3. utility
3. Use CSS imports to import the files
4. Don't forget to test by refreshing the browser as you go to see if anything broke

15 mins

Exercise 6b: building with grunt

1. Install node and the grunt cli
   nodejs.org/download/
   npm install -g grunt-cli

2. Copy .gitignore file from part06b files

3. Copy package.json file from part06b files

4. Run: npm install

5. Copy Gruntfile.js file from part06b files

6. Run: grunt

7. Update reference to main.min.css in index.html

Just a bit more CSS

Table specific CSS

- border-collapse
  - borders by default in tables are separated and will stack
  - values: collapse | separate | inherit
- border-spacing
  - sets how much space appears between borders
- table-layout
  - defines algorithm used to calculate table layout
  - values: auto (default) | fixed
- Separate with spacing demo
- Striped rows demo
- table-layout fixed demo 1 and demo 2

```
table {
  border-collapse: collapse;
  border-spacing: 0px;
  table-layout: fixed;
}
```

45 mins

Exercise 7a: baseline table

1. Install Grunt watch
   https://github.com/gruntjs/grunt-contrib-watch
   1. Configure it to watch your css changes
2. Hide the form area & unhide the surgeon merge area
3. Use your grid system to create a 2 column layout for your 2 tables (surgeon list & merge area)
4. Style the right most table (basic table) including the button bar (gray)

Tip: Use positional selectors

30 mins

Exercise 7b: accordion table

1. With your baseline table established, build out the left table design. Call this one “accordion”

2. Add vertical borders between tables

3. **Extra credit**: see if you can get the tables to align properly. The right table’s line should be aligned to bottom of header on left table

20 mins

Exercise 7c: the rest

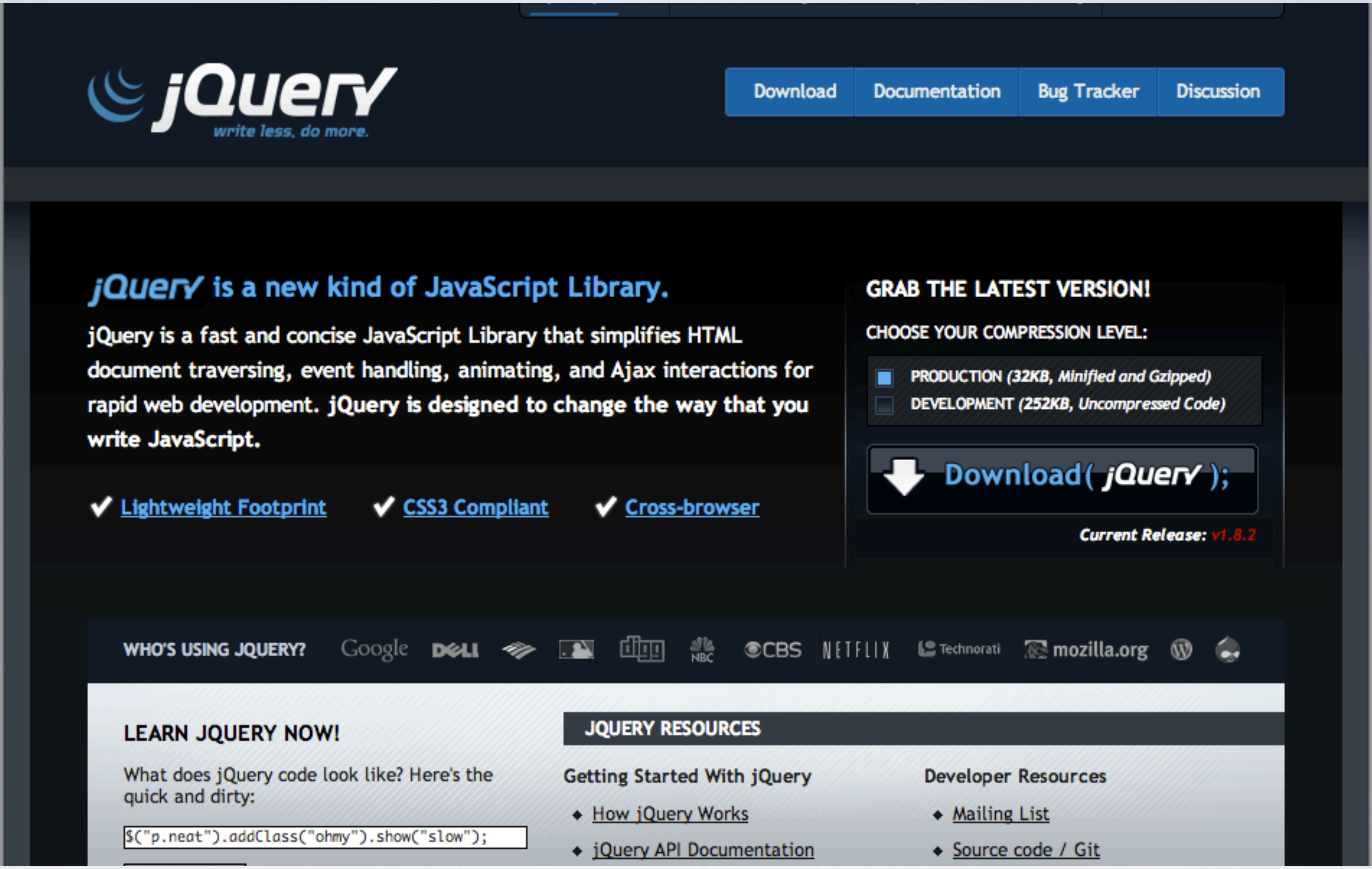
- Style the “manage duplicate surgeons" content area
- Style the filter form
  Tip: use inline block

JavaScript

jQuery

jQuery

- Free & open source JavaScript library
- Used primarily for HTML:
- DOM Traversal
- Event handling
- Ajax interactions
- Animations
- Highly extensible



jQuery is a new kind of JavaScript Library.

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.

✓ Lightweight Footprint ✓ CSS3 Compliant ✓ Cross-browser

GRAB THE LATEST VERSION!

CHOOSE YOUR COMPRESSION LEVEL:

[✓ PRODUCTION *(32KB, Minified and Gzipped)*]
DEVELOPMENT *(252KB, Uncompressed Code)*

Current Release: v1.8.2

LEARN JQUERY NOW!

What does jQuery code look like? Here's the quick and dirty:

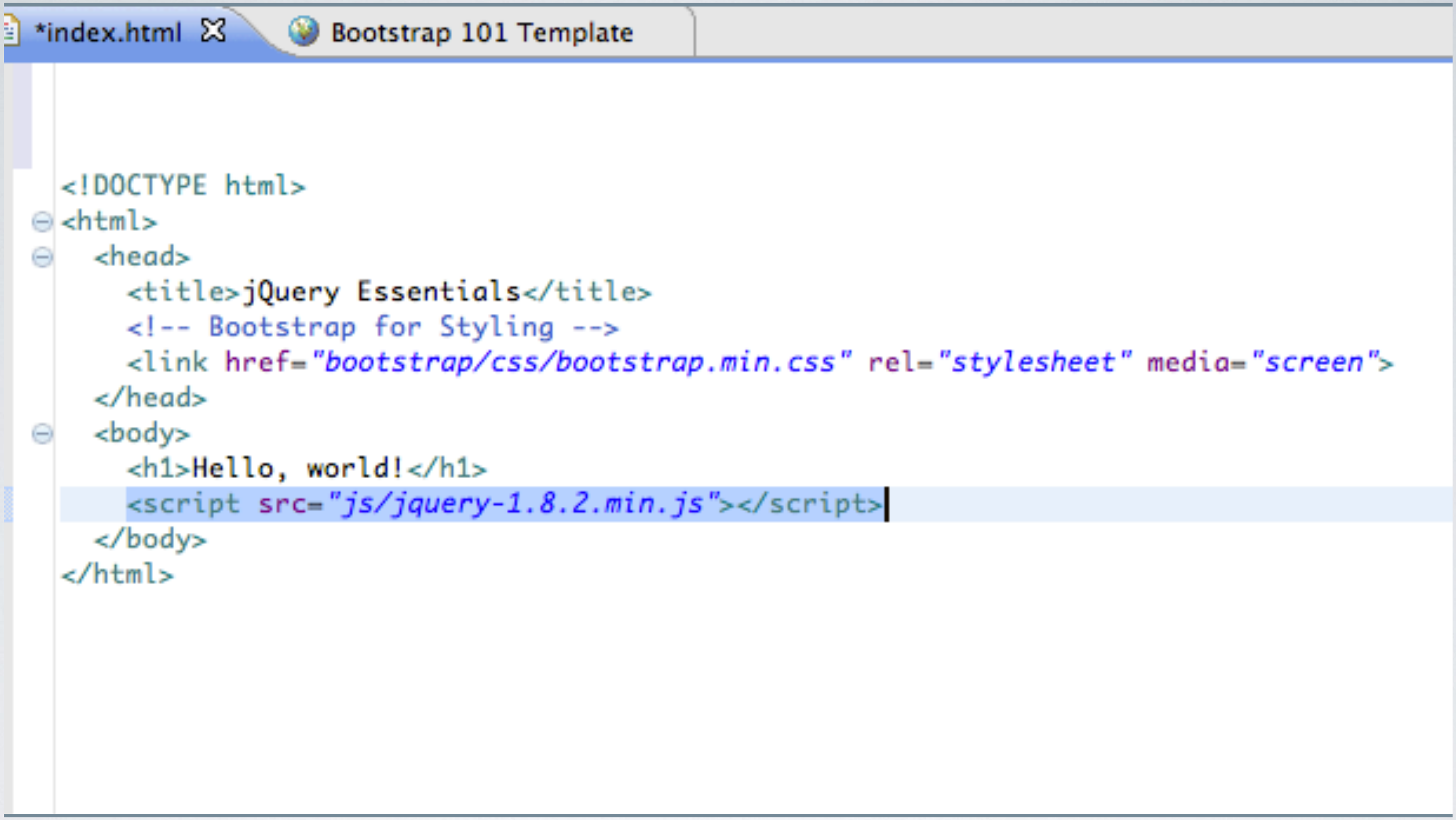
```
$("p.neat").addClass("ohmy").show("slow");
```

JQUERY RESOURCES

Getting Started With jQuery
- How jQuery Works
- jQuery API Documentation

Developer Resources
- Mailing List
- Source code / Git

Installing jQuery



```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Essentials</title>
    <!-- Bootstrap for Styling -->
    <link href="bootstrap/css/bootstrap.min.css" rel="stylesheet" media="screen">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="js/jquery-1.8.2.min.js"></script>
  </body>
</html>
```

Installing jQuery

Wiring up a Page

```
$(document).ready(function () {
  // the page has loaded
  // wrap all code in this
  myApp.init();
});

$(function() {
  // equivalent
});
```

Selectors in jQuery

```
/* selects all <form> elements on the page */
form { ... }

/* selects the first <li> on the page */
li:first-child { ... }

/* selects element with id="my-id" */
#my-id { ... }

/* selects all elements containing class "my-
class" */
.my-class { ... }
```

```
/* selects all <form> elements on the page */
$("form");

/* selects the first <li> on the page */
$("li:first-child");

/* selects element with id="my-id" */
$("#my-id");

/* selects all elements containing class "my-
class" */
$(".my-class");
```

Selectors in jQuery

```
$("form input");


$("ul li:first-child");









$("#my-form input:checked");


$("#my-id .js-my-class span");
```

```
$("form input");


$("ul li:first-child");
/* or...*/
$("ul").find("li:first-child");
/* or...*/
$("ul").find("li").eq(0);




$("#my-form").find(”input:checked”);


$("#my-id").find(".js-my-class")
     .find("span");
```

Tip: Caching & Chaining

```
/* Bad: no caching or chaining */
$("#my-input").show();
$("#my-input").css("left", "10px");
$("#my-input").addClass("error");
$("#my-input").val("There was an error in this field");
```

Tip: Caching & Chaining

```
/* Caching jQuery selectors into variables */
var $myInputVar = $("#my-input");
$myInputVar.show();
$myInputVar.css("left", "10px");
$myInputVar.addClass("error");
$myInputVar.val("There was an error in this field");
```

Tip: Caching & Chaining

```
/* Chaining */
$("#my-input").show()
     .css("left", "10px")
     .addClass("error")
     .val("There was an error in this field");
```

Quiz

How many elements will return from the following statement:?

```
<ul id=“my-list”>
     <li>First</li>
     <li>Second</li>
     <li>Third</li>
</ul>
<script>
     $("#my-list li:first-child”);
</script>
```

A. None
B. One element
C. Two elements
D. Three elements

Quiz

How many elements will return from the following statement:?

```
//Assume we observe the rule of 1 element per ID
$(".my-list #my-item”);
```

A. None
B. One element
C. Two elements
D. Three elements

Quiz

How many elements will return from the following statement:?

```
//Assume we observe the rule of 1 element per ID
$(“#my-list").find("li:first-child").eq(1);
```

A. None
B. One element
C. Two elements
D. Three elements

Manipulating the DOM

.html() as a getter

```html
<div class="js-message info">
     <strong>Hello</strong> World
</div>
```

```javascript
val = $(".js-message").html();

/*
 val = “<strong>Hello</strong> World”
*/
```

.html() as a setter

Setting value with .html()

```
$(".js-message")
     .html(“<span>Check</span>
          this out”);
```

Yields this…

```
<div class="js-message">
     <span>Check</span> this out
</div>
```

.prop() as a getter

```
<body>
     <input type="checkbox" checked="checked" class="js-test first" />
     <input type="checkbox" class="js-test second" />

     <script>
          var isChecked = $(“.js-test”).prop("checked");

          /* isChecked = true */
          var classVal = $(".js-test").prop("class");
          /* classVal = "js-test first" */
     </script>
</body>
```

```
/* Alternative approach for testing checked state a given selector: */
var isChecked = $(“.js-test”).is(":checked");
// isChecked = true
```

.prop() as a setter

```
<body>
     <input type="checkbox" checked="checked" class="js-test" />
     <input type="checkbox" class="js-test" />
</body>
```

```
$(".js-test").prop(”checked", true).prop("disabled", true);

/* would check and disabled both checkboxes */
```

prop() & val() on form elements

Setting value with .html()

```
$(“#name”).prop(“value”, “the value”);

/* or .val() do the same thing */
$(“#name”).val(“the value”);



/* this would return “the value” as a string
*/
$(“#name”).val();
```

```
<input type=“text” id=“name” value=“the
value” />
```

data()

```
<!-- Sets data-foo = "bar" on <body> ->
<body data-foo="" data-state=“loading">
<!-- contents of page goes here ->
</body>
```

```
//Sets data-foo = "bar" on <body>
$("body").data("foo", "bar");

//returns “bar"
$("body").data("foo");
```

mo' data()

```
//sets data-foo to object literal
$("body").data("foo", {
    "name": "Eric",
    "title": "Lead front-end developer",
    "isEmployed": true
});

//returns "Lead front-end developer”
$("body").data("foo").title;

//returns true
$(“body").data("foo").isEmployed;
```

addClass()

```
<div class=“js-message”>
     Check this out
</div>
```

+

```
$(“.js-message”).addClass(“error”)
```

↓

```
<div class=“js-message error”>
     Check this out
</div>
```

removeClass()

```
<div class=“js-message error”>
     Check this out
</div>
```

+

```
$(“.js-message”).removeClass(“error”)
```

```
<div class=“js-message”>
     Check this out
</div>
```

Example: Validation

```
//cache the username and error message elements
var $username = $(”#username”),
    $errorMessage = $(”#errorMessage”);

if( $.trim( $username.val() ).length === 0 ){
     $username.addClass(“error”);
     $errorMessage.show().text(“Enter a username”);
} else {
     $username.removeClass(“error”);
     $errorMessage.hide();
}
```

Displaying data (poor performance)

```
<ul></ul>
```

```
//create an array and cache DOM list where we will output list
var fruit = [“Apples”, “Oranges”, “Pears”],
    $list = $(”ul”);

//Loop through array and append html snippet to the <ul>
for ( var i = 0, len = fruit.length; i < len; i++ ){
    $list.append(“<li>” + fruit[i] + “</li>”);  //works but bad for performance
}
```

```
<ul>
   <li>Apples</li>
   <li>Oranges</li>
   <li>Pears</li>
<ul>
```

Displaying data (better performance)

```
//create an array and cache DOM list where we will output list
var fruit = [“Apples”, “Oranges”, “Pears”],
    $list = $(”ul”),
    htmlSnippet = ””;

//Loop through array and append html snippet to the <ul>
for ( var i = 0, len = fruit.length; i < len; i++ ){
     //append html string to local variable
     htmlSnippet = htmlSnippet + “<li>” + fruit[i] + “</li>”;
}

$list.append( htmlSnippet );  //only touches the DOM once
```

Quiz

What does **result** equal

```
<div id="summary" data-foo="bar" foo="baz"></div>
<script>
    var result = $("#summary").data("foo");
</script>
```

A. "baz"
B. "bar"
C. "barbaz"
D. None of the above

Quiz

What does this expression return?

```
$("#summary").addClass("error warning message")
     .removeClass("warning").prop("class", "no-error").prop("class");
```

A. $("summary")
B. "error message no-error"
C. "no-error"
D. null

Event Handling

Types of events

- Mouse
  - (up, down, move)
  - click, dblclick, hover
- Mouse
  - (over, out)
- Key, generally form elements
  - (up, down, press)
- Form
  - (blur, focus)
- Window (only)
  - (resize, scroll)

```
<a id="link">Link</a>


<script>
$("#link")
  .on("click", function(e) {
    console.log("fired on down and up");
  })
 .on("focus", function(e) {
    console.log("key presses go to me");
  });
</script>
```

Events & their relationship

http://codepen.io/ericmasiello/pen/FotDc

Event handlers

- The event handler should have a name and not be an anonymous
- The this keyword provide access to the DOM object the event was registered to
- Any object can have an unlimited number of event handlers attached to it
  - Make sure you manage this
  - The off method removes event handlers

```
var $link = $(“#link”);
var link_onClick = function(e) {
  // is true
  $(this).is($link);
};

$link.off(“click”);
$link.off(“click”, link_onClick);
$link.on(“click”, link_onClick);
```

Key presses

- The key events are *mostly* similar to mouse events
  - mouseup === keyup
  - click === keypress
  - mousedown !== keydown
    - mousedown fires once
    - keydown fires continuously
- Key events will not fire until the object you attach them to has the focus
- Use window for global key press events

```
var submit_onKeyUp = function(e) {
  // find form by traversing up the tree
  var $form = $(this).closest(“form”);
  if (e.keyCode === 13) {
    /* pressed enter, submit */
    $form.submit();
  }
};
$(“input.searchbox”)
  .on(‘keyup’, submit_onKeyUp);
```

Bubbling

- We can achieve a huge performance gain by applying event listeners to parents if:
    - the children are similar
    - the chidren generally will have the same event happen to them
    - e.g. list items and table rows
- This is called bubbling

```
<table>
     <tr>…</tr>
     …
     <tr>…</tr>
</table>

<script>
var tableRow_onHover = function(e) {
  // this is true
  ($(this).prop("tagName") === “tr”);
};

// one event listener, instead of n rows
$(“table”)
 .on(“hover”, “tr”, tableRow_onHover);
</script>
```

Default behavior

- We can override functionality three ways
  - **e.preventDefault()**
    - Stops default behavior
    - http://codepen.io/ericmasiello/pen/txKLI
  - **e.stopPropagation()**
    - Prevents event from bubbling
    - http://codepen.io/ericmasiello/pen/ruety
  - **returns false**
    - Fires both preventDefault() and stopPropagation()
    - http://codepen.io/ericmasiello/pen/gDABv

Quiz

What happens to any <a> when clicked on?

```
$(“.links”).on(“click”, “a”, anchor_onClick);


var anchor_onClick = function(e) {
  $(e).css(“color”, “red”);
};
```

A. Text color turns red
B. Background color turns red
C. Runtime error
D. Only responds to <a> with class=“links”

Quiz

What is the value of c after clicking on .woodchuck?

```
var c = 0;

$(“input.woodchuck”).on(“keydown”, function(e) { c++; });
$(“input.woodchuck”).on(“keyup”, function(e) { c++; });
$(“input.woodchuck”).on(“keypress”, function(e) { c++; });
```

A. c === 2 (three equals uses strict type, always use)
B. c === 3
C. Indeterminate
D. Syntax error

Quiz

What is needed to prevent the div event handler from firing?

```
<div><span> a </span></div>
<script>
     $(‘div’).on(‘click’, function(e) { /* does stuff */ });
     $(‘span’).on(‘click’, function(e) {
          /* insert here */
     });
</script>
```

A. e.stopPropagation();
B. e.preventDefault();
C. return false;
D. Both A and B
E. Either A or C
F. Use bubbling on the div instead

45 mins

Exercise 08a: basic jQuery

1. Install jshint and add to Grunt
   https://github.com/gruntjs/grunt-contrib-jshint
2. Copy .jshintrc file from assets folder
3. Use strict mode in all your JavaScript files "use strict";
   http://www.w3schools.com/js/js_strict.asp
4. Inside main.js
   1. Add click events for 2 main tabs that hide/show major sections of the application
   2. Add the "is-selected" class to the selected tab
5. Create account.js
   1. Create the "Same as cell number" functionality on click of the checkbox. It needs to copy the values from cell phone and disables the home number fields
   2. Put new password into an error state (CSS) if values don't match

Ajax

$.ajax and callbacks

```
$.ajax({
     "type": "GET",
     "url": "../php/api.php",
     "contentType": "application/json; charset=utf-8",
     "dataType": "json"

     }).done( function( data, textStatus, jqXHR ) {
          //Successful response here
     }).fail( function( jqXHR, textStatus, errorThrown) {
          //Failed response here
     }).always( function() {
          //Always executes after .done() or .fail()
     });
```

Saving data

```
$.ajax({
     /* or could be a PUT */
     "type": "POST",
     "url": "../php/api.php",
     "contentType": "application/json; charset=utf-8",
     "dataType": "json",
     "data": {
          "id": 5,
          "firstName": "Bob",
          "lastName": "Jones"
     }
}).done( function( data ) {
     //Successful response here
});
```

Managing multiple ajax requests

```
$.when(
    //ajax call 1
    $.ajax().done( function(){} ),
    //ajax call 2
    $.ajax().done( function(){} ),
    //ajax call 3
    $.ajax().done( function(){} )

}).done( function( response1, response2, response3 ) {


    //Executes once all ajaxCalls have successfully returned
    //response1[0] = data
    //response1[1] = textStatus
    //response1[2] = jqXHR
});
```

Quiz

What does result equal once everything has executed?

```
result += "hello ";
$.when(

    $.ajax({"url": "api.php"})
         .always( function(){ result += "Bob "; } )
         .done( function(){ result += "Jim "; } )

}).done( function() {
    result += "Eric ";

});

result += "Tim ";
```

A. hello Bob Jim Eric Tim
B. hello Tim Bob Jim Eric
C. hello Tim Jim Bob Eric
D. hello Jim Bob Eric Tim

Questions?

30 minutes

Exercise 08b: ajax

1. Copy sampledata.json from assets folder into your project
2. Load sampledata.json using an ajax get request
3. Render the JSON data as HTML and append to the surgeons table.
    1. Use this as your template: http://codepen.io/ericmasiello/pen/yzcCg
    2. You'll also need to update state.css for the is-warning class

So this kinda sucks

- functions are everywhere
- everything is unstructured
- everything is in the global name space (window object)
- inline templates in JS is difficult to maintain
- if you refresh your browser, it doesn't retain which tab you had open

Routing with Sammy

- http://sammyjs.org/

```
Sammy(function () {

  // If no matching path is found
  this.notFound = function (){
    // Will reroute to the manager tab
    document.location.href = '#!/manager';
  };

  // matches yoursite/#!/manager
  this.get('#!/manager', function () {
    toggleTabs('manager');
  });

  // matches yoursite/#!/manager
  this.get('#!/account', function () {
    toggleTabs('account');
  });
}).run();
```

20 minutes

Exercise 08c: routing

1. Copy sammy.js from assets into your js/vendor folder
2. Add reference to sammy.js in your index.html file
3. Replace code in main.js with this: http://codepen.io/ericmasiello/pen/sDgBh

Some JavaScript basics

JavaScript has 2 types

- **Primitives**: stored as simple data types
    - **Boolean**: true or false
    - **Number**: Any integer or floating point numeric value
    - **String**: Any character sequence delimited by single or double quotes
    - **Null**: a primitive type that has only one value, null
    - **Undefined**: a primitive type that has only one value, undefined. It is the value assigned to a variable that is not initialized. e.g. var foo; (foo === undefined)
- **References**: more on these later…

Primitives

- use typeof to identify
- except for null…
- for null use === null
- Demo

```
var str = “string”;
var bool = true;
var num = 2;
var undef;
var n = null;

typeof str === “string”; // true;
typeof bool === “boolean”; // true;
typeof num === “number”; // true;
typeof undef === “undefined”; // true;
typeof n === “null”; // false; “object”
n === null; // true;
```

Comparison without coercion

- Use === and !==

```
“1” == 1; /* true; double equals converts
string to number */
“1” === 1; // false; different type

undefined == null; // true;
undefined === null; // false
```

References

- Stored as objects which are really just references to locations in memory
- Object is an unordered list of properties consisting of a name (as a string) and a value (you can think of as nothing more than hash tables)
- When a value of a property is a function, its considered to be a method
- Functions in Javascript are also reference values
- References do not store the object directly into the variable to which they are assigned. Instead it holds a pointer (reference) to the location in memory. This is the primary difference between reference and primitive values

Creating / “instantiating” objects

- Use *new* keyword with a *constructor*
- Any function can be a constructor
- By convention, constructors start with capital letters

```
function Claim(){
  //does stuff
};


var claim = new Claim();
```

Since objects are references…

```
var obj1 = new Object();
var obj2 = obj1;

//set property on obj2
obj2.myCustomProperty = “hi”;

//therefore obj1 has that property
obj1.myCustomProperty === “hi”; //true

//dereference objects by setting them to null
obj2 = null;

obj1.myCustomProperty === “hi”;  //true
obj2 === null;  //true
```

Creating built in reference types

- You can use the *new* keyword
- Some have a **much preferred** literal form

```
var x = new Object();
var x = new Array();
var x = new Date();
var x = new Function();

//Literal forms…

var x = {
  “key”: “value
};
var x = [“banana”, “apple”, “pear”];

//no literal form of Date

function x(){

  //do stuff

};
```

Functions

- Function declarations
- Function expressions
  - Store references to the functions
  - Can then be passed around like any other variable
  - Can also be invoked immediately (IIFE)

```
//declaration
function plus(val1, val2){
  return val1 + val2;
}

plus(1, 2) === 3;

//expression
var plus = function(val1, val2){
  return val1 + val2;
};

plus(1, 2) === 3;

//IIFE = Immediately invoked function expression
var plus = function(val1, val2){
  return val1 + val2;
}(2, 3);
plus === 5;
```

Scope

- Anything declared with a var is scoped to a function, not a block
- **let** is a new way to define variables in ES6 but isn't available to most browsers yet
- Variables defined within the scope of a function are hoisted to the top of the function and defined there, however their value won't be set yet (undefined)
- If you omit the var keyword, the variable will be considered global (and therefore a property of the window object) which can be bad… "use strict" will block you from being able to use variables not declared with var;

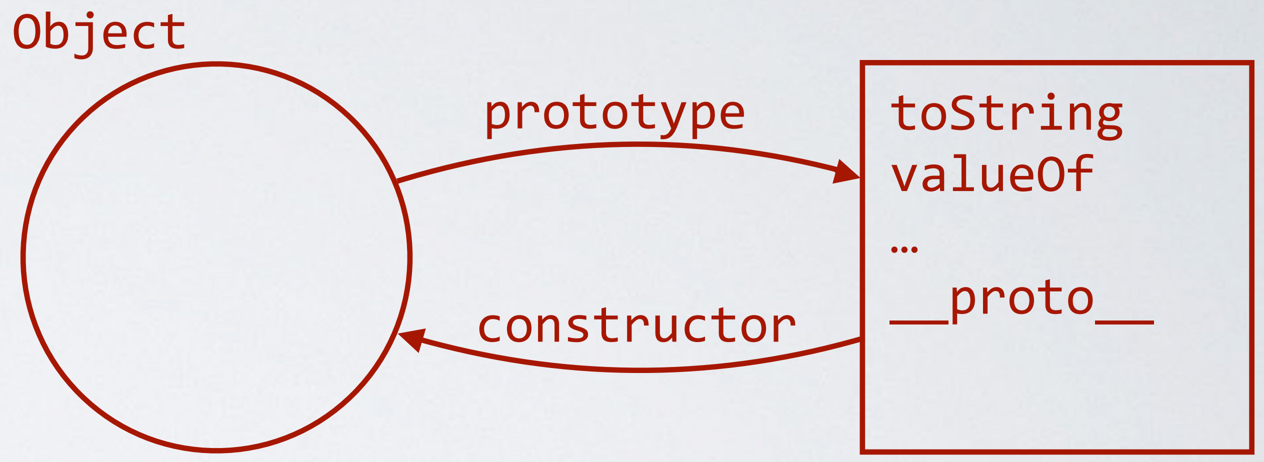
this

- Value of this within a function is determined by how the function was called
- Traditional invocation Demo
- Functions as methods Demo
  - methods are just properties of objects
- *new* and constructor Demo
- call, apply, and bind Demo
  - call and apply effectively do the same thing - allow you to specify the "this" value at call time
  - bind is used to specify the this value when the function is defined

prototype

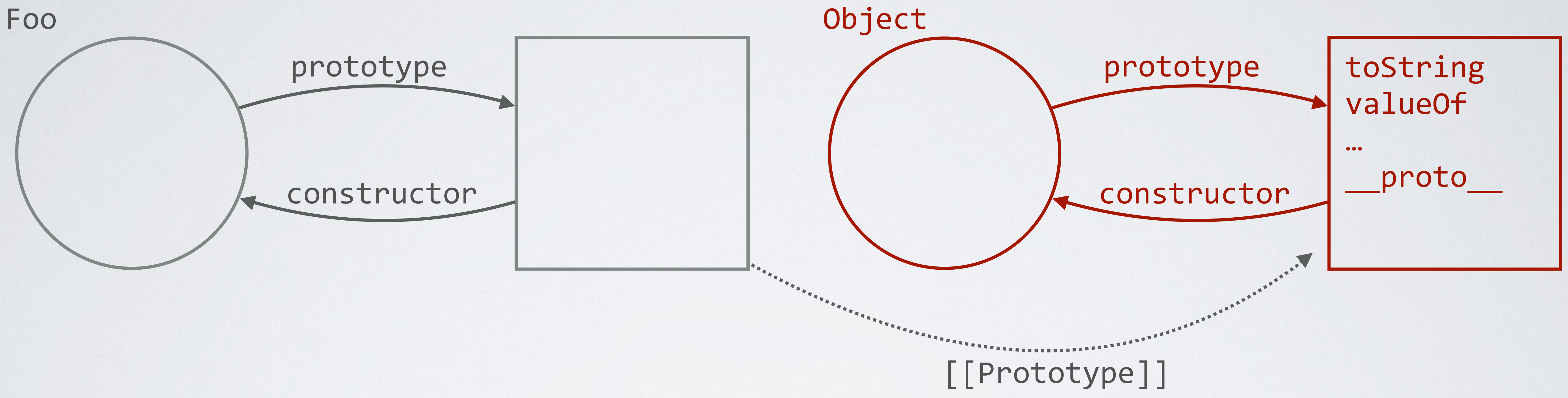
- Constructors create objects that are linked to the constructor's prototype
- Allow you extend the functionality of a object to all instances
- Demo

prototype and *new*



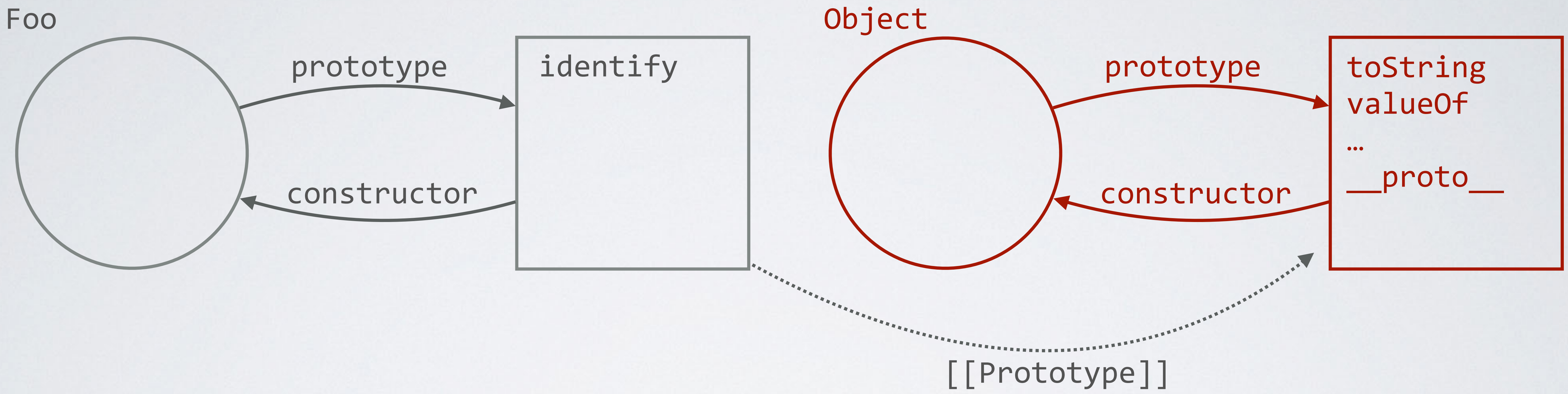
http://codepen.io/ericmasiello/pen/hoJGk?editors=001

prototype and *new*



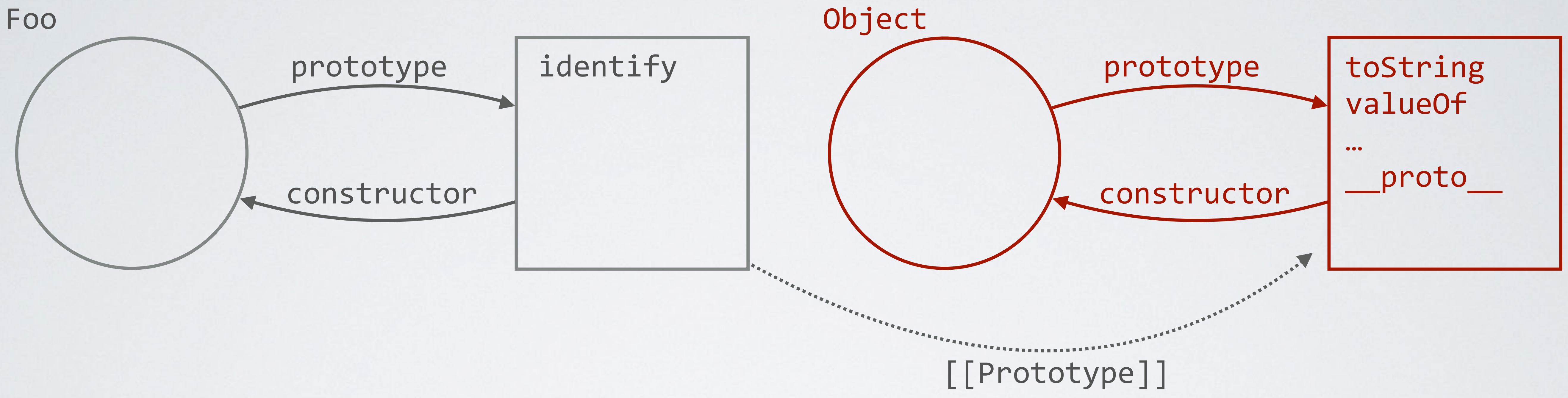
http://codepen.io/ericmasiello/pen/hoJGk?editors=001

prototype and *new*



http://codepen.io/ericmasiello/pen/hoJGk?editors=001

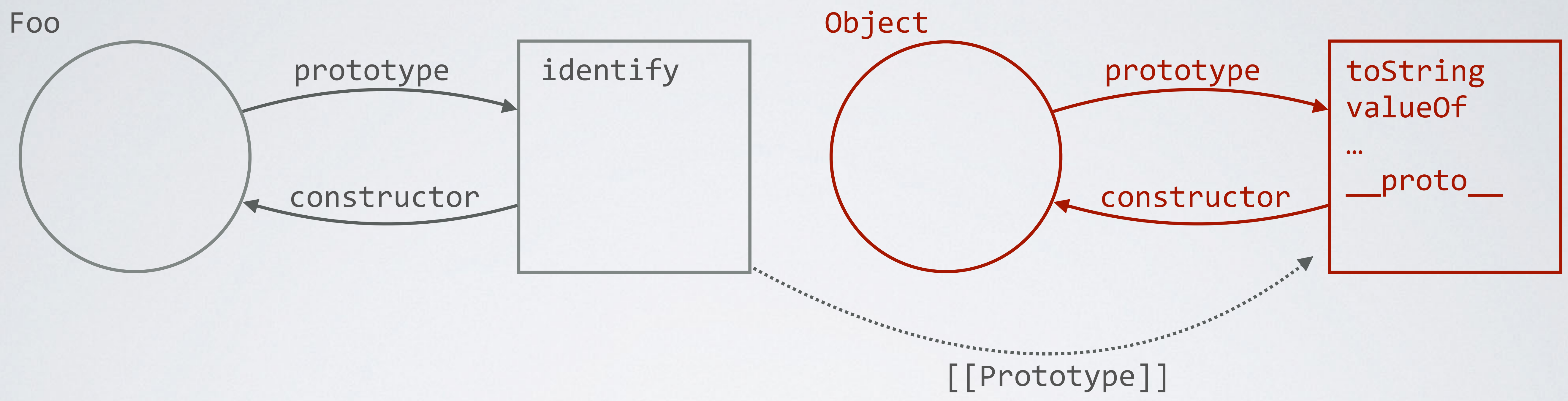
prototype and *new*



```
var a1 = new Foo(“a1”);
```

http://codepen.io/ericmasiello/pen/hoJGk?editors=001

prototype and *new*

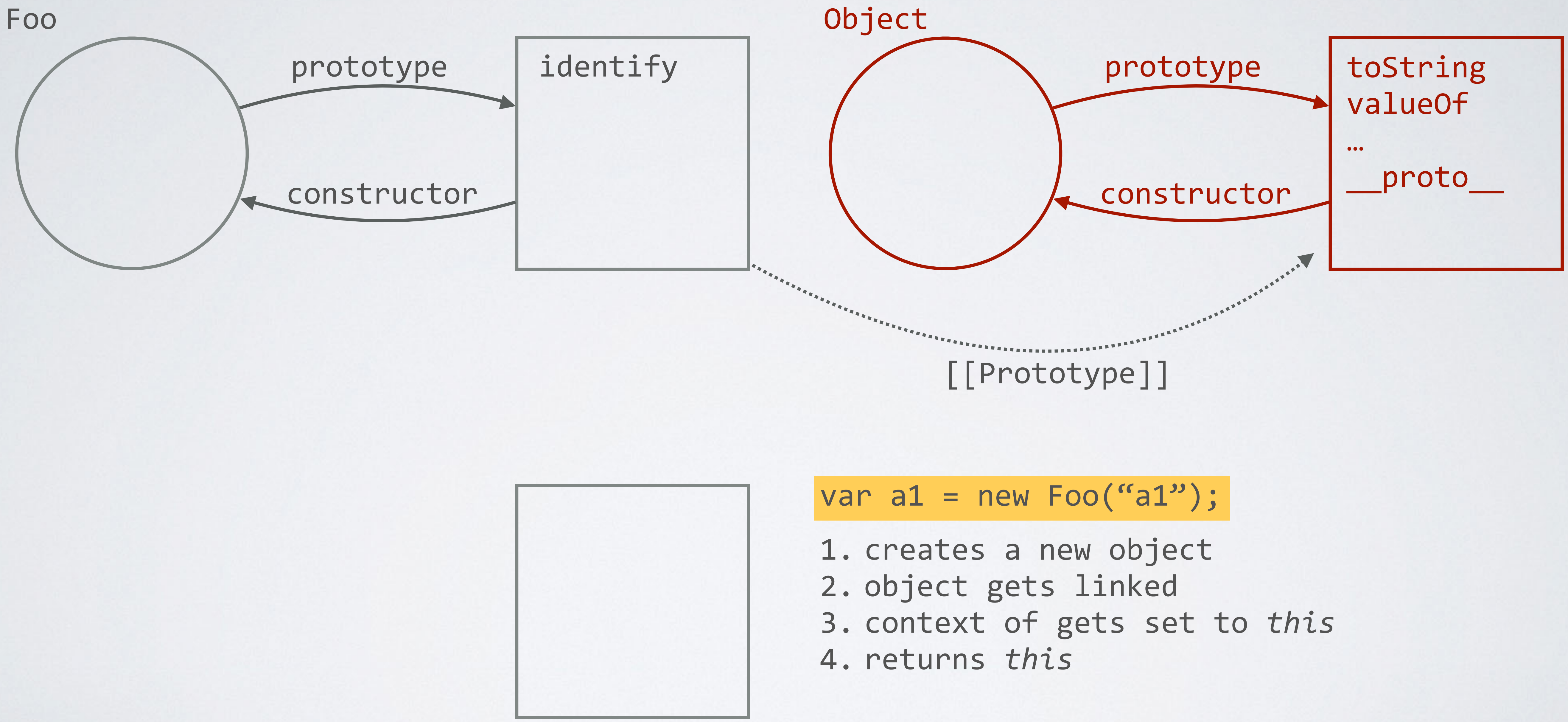


```
var a1 = new Foo("a1");
```

1. creates a new object
2. object gets linked
3. context of gets set to *this*
4. returns *this*

http://codepen.io/ericmasiello/pen/hoJGk?editors=001

prototype and *new*

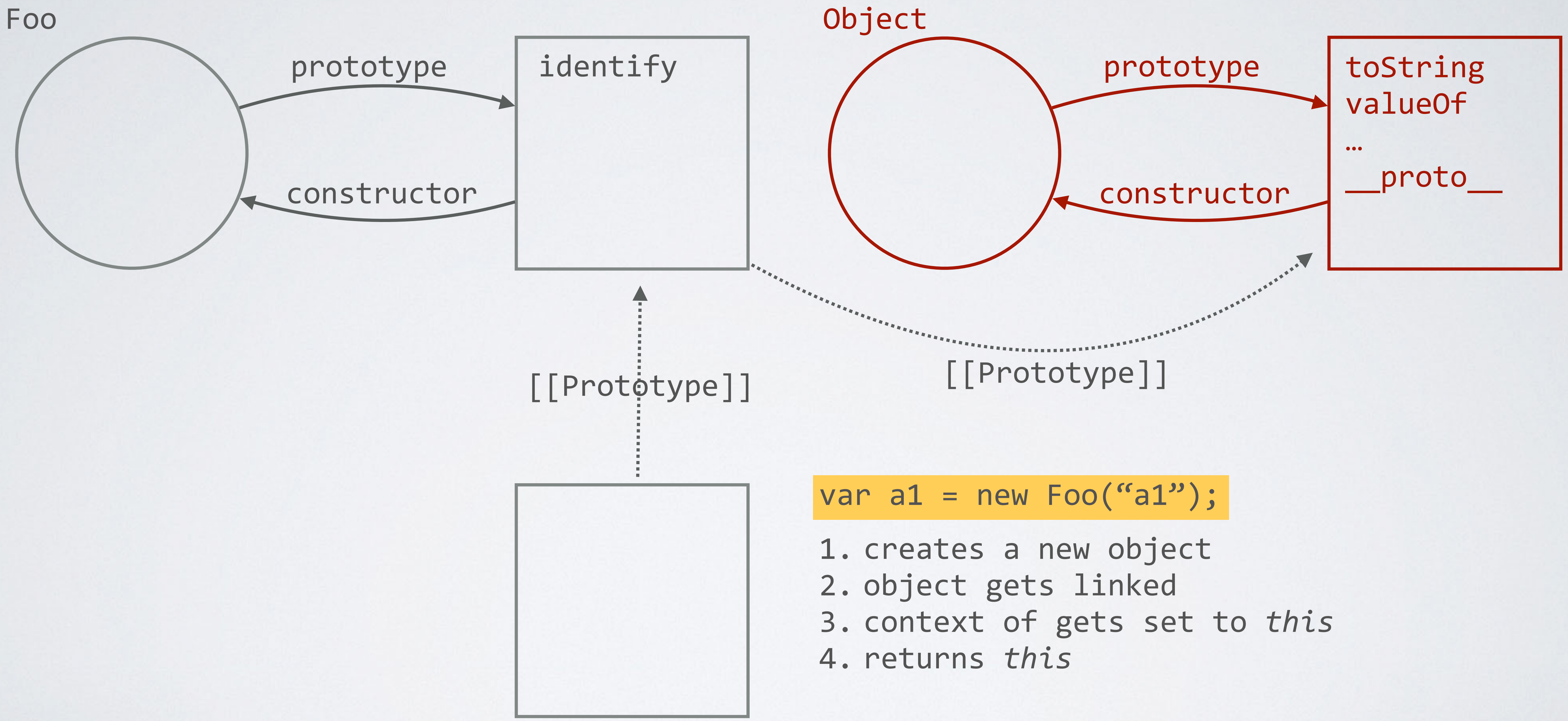


```
var a1 = new Foo("a1");
```

1. creates a new object
2. object gets linked
3. context of gets set to *this*
4. returns *this*

http://codepen.io/ericmasiello/pen/hoJGk?editors=001

prototype and *new*

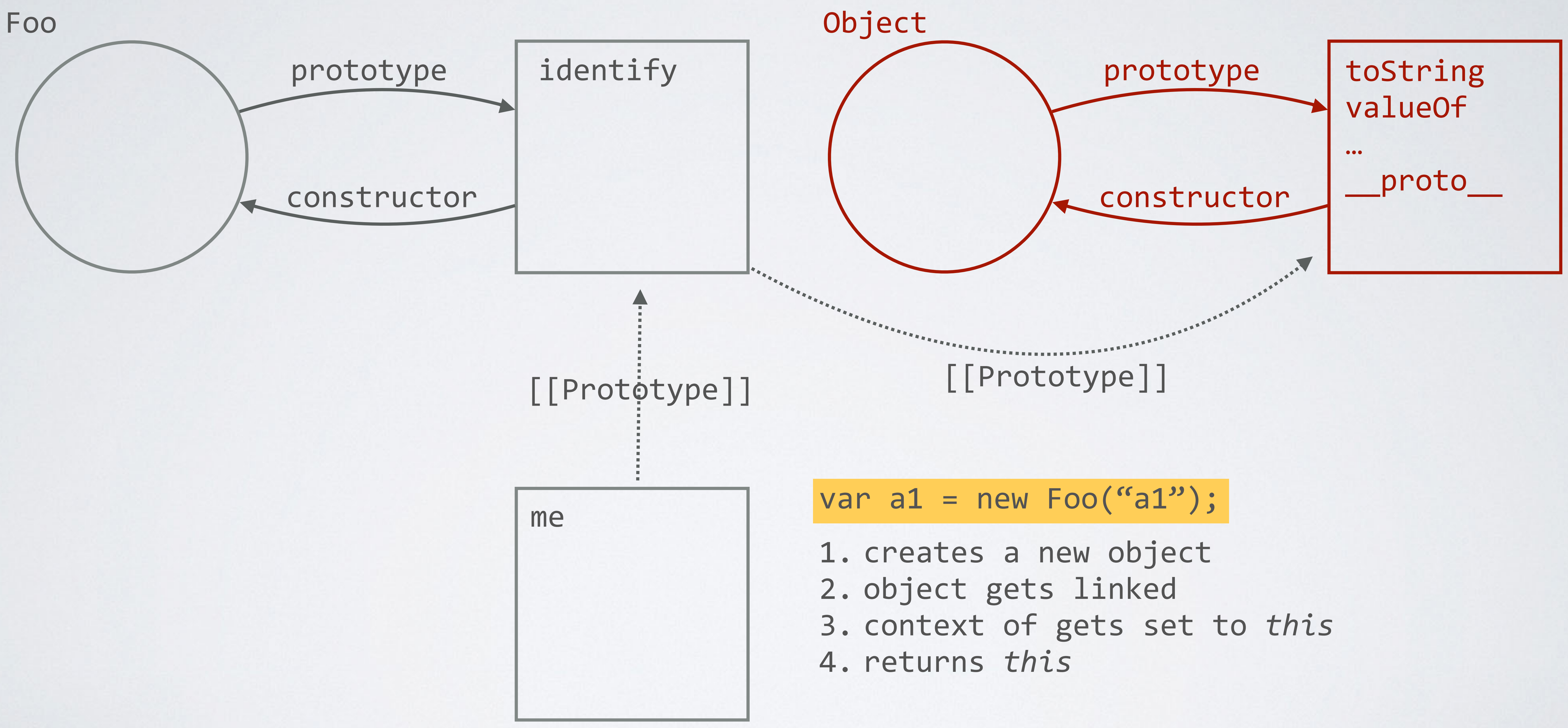


```
var a1 = new Foo("a1");
```

1. creates a new object
2. object gets linked
3. context of gets set to *this*
4. returns *this*

http://codepen.io/ericmasiello/pen/hoJGk?editors=001

prototype and *new*

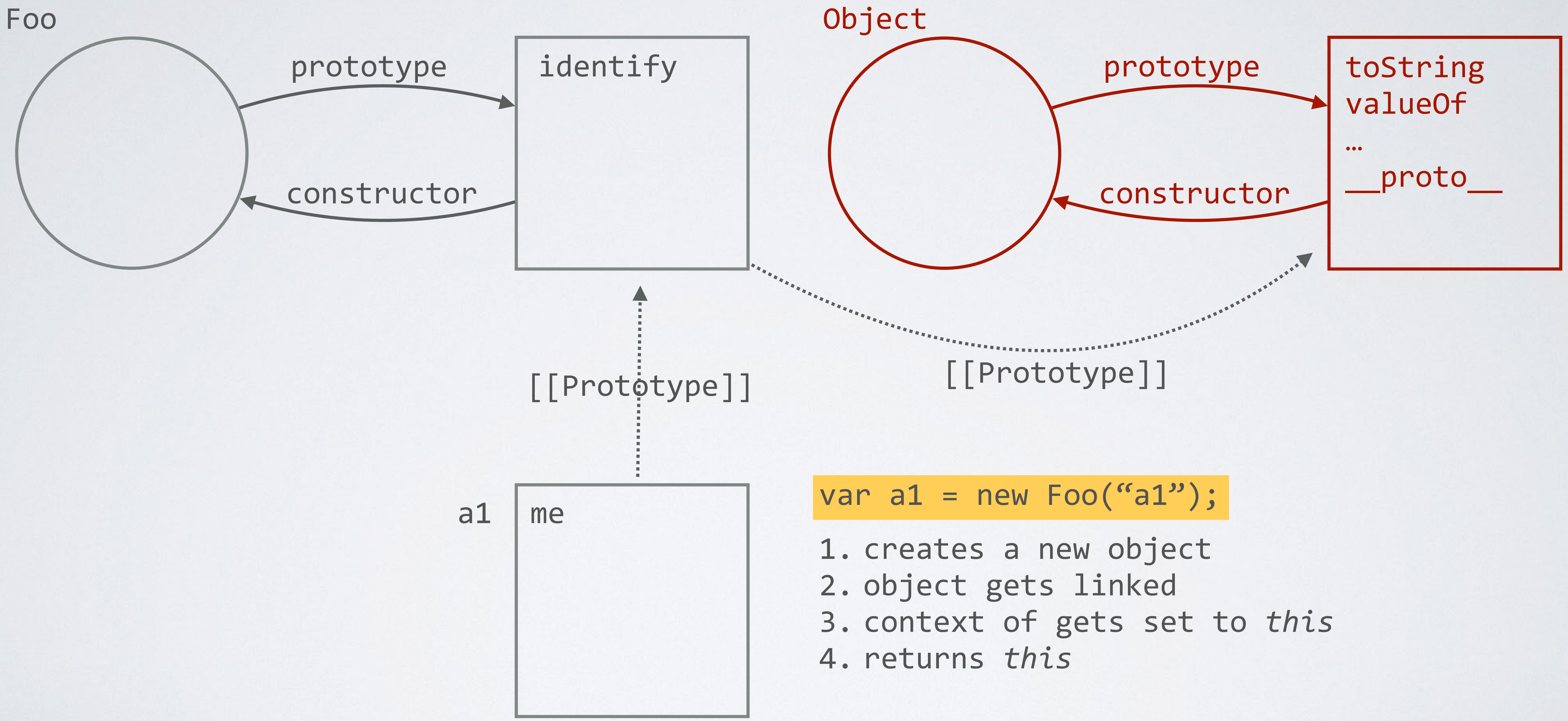


`var a1 = new Foo("a1");`

1. creates a new object
2. object gets linked
3. context of gets set to *this*
4. returns *this*

http://codepen.io/ericmasiello/pen/hoJGk?editors=001

prototype and *new*

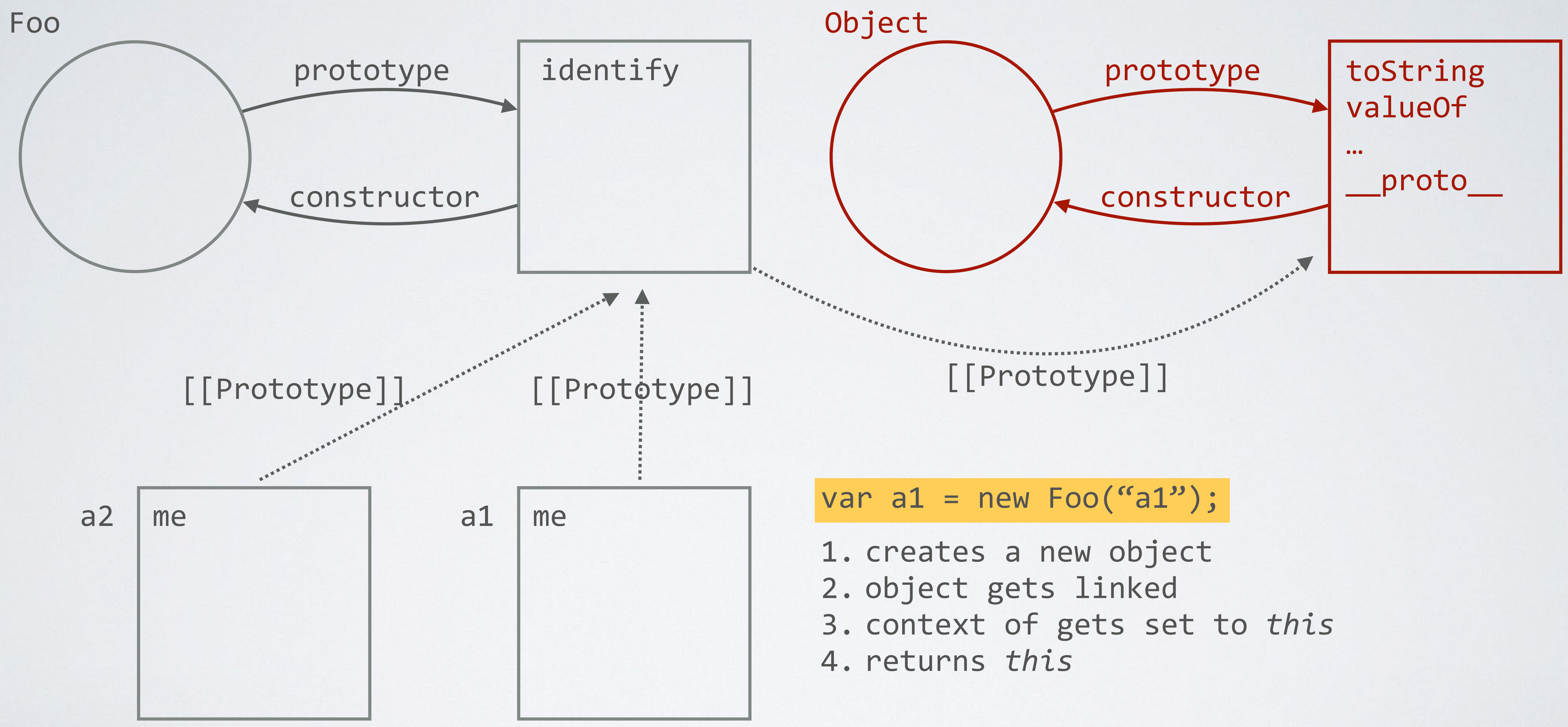


```
var a1 = new Foo("a1");
```

1. creates a new object
2. object gets linked
3. context of gets set to *this*
4. returns *this*

http://codepen.io/ericmasiello/pen/hoJGk?editors=001

prototype and *new*



```
var a1 = new Foo("a1");
```

1. creates a new object
2. object gets linked
3. context of gets set to *this*
4. returns *this*

http://codepen.io/ericmasiello/pen/hoJGk?editors=001

Module pattern

- Must have an outer enclosing function to keep everything encapsulated
- Wrap code in an IIFE (immediately invoked function expression) http://benalman.com/news/2010/11/immediately-invoked-function-expression/
- Must return at least one function
    - Expose public methods via return an object that references those private methods
- require.js or AMD are extensions of the classic module pattern
- Demo

20 mins

Exercise 9: modularize js

- Use the modular pattern to wrap your the surgeons, account, and main.js code into modules that only expose their init methods publicly
- Return the main.js module as "app"

Knockout JS

http://knockoutjs.com/documentation/introduction.html

Knockout JS

- Elegant dependency tracking: automatically updates the right parts of your UI whenever your data model changes.
- Declarative bindings: a simple and obvious way to connect parts of your UI to your data model.
  - Observables
  - ObservableArrays
  - Computed Observables
    - Observables that are based on the value of other observables.
    - Example: you want to display full name which is based on two other observables first name + last name
- Templates
- Demo

20 mins

Exercise 10a: intro to KO

- As a class
  - Walk through the following example: http://codepen.io/ericmasiello/pen/LCrKv?editors=101
- Copy Knockout library from assets folder to your project
  - Update your js and html with our example code above
  - Replace document.ready with ko.applyBindings

30 mins

Exercise 10b: KOnverstion

- Replace your jQuery code with KO to:
  - Manage hiding/showing the #manager and #account sections of the page with KO observable(s)
  - Handle account form input validation and copying cell phone into home phone

30 mins

Exercise 10c: more ajax

- Copy sampleuser.json & KO mappings plugin from assets to your project

- Use Knockout mappings plugin to initialize observables to the same names as the properties in sampleuser.json.
  http://knockoutjs.com/documentation/plugins-mapping.html
  Hint: You'll need to add other custom observables e.g. newPassword and newPassword2

- Load data using ajax and apply data to observables using Knockout mappings plugin

Communicating via pub sub

- Publish and subscribe to custom event notifications
- Provides a way to communicate between two modules that don’t have any direct interfaces with one another
- Simple demo
- Parent-child pub sub model demo

30 mins

Exercise 10d: merger module

1. Copy the pubsub.js from the assets folder to your vendor folder

2. Copy sampledata2.json from assets to your project

3. Add PubSub to the globals object in .jshintrc

4. Review merger module (copy to your project)
   http://codepen.io/ericmasiello/pen/vGwuc?editors=001

5. When the user clicks "Manage Duplicates" you need to pass data to the merger module. Use pub sub to accomplish this.

6. When the user clicks the "Confirm Duplicates" button, the callback publishes an event called 'spc/merger/merged-record'. Create a subscription to this event that will reload the surgeon list calling sampledata2.json.

45 mins

Exercise 10e

1. Display an error icon whenever there is a possible duplicate. Look for the possibleDuplicate flag on each surgeon record
2. Update surgeon templates to support different states:
    1. Once you click Manage Duplicates, the selected item should change to "selected" and every other surgeon's link should change to "Add to list" with a + icon
    2. When you click "Add to list" the text should change to "Added" with a green check
    3. You'll need one more icon (dark right arrow) when in the "Selected" state (in assets folder)
    4. Reset the state on the surgeon list when user clicks Cancel on the merge list
3. Collapse/expand toggle ability
    1. Display the correct template for grouped items
    2. Update template for outputting the child rows
    3. If you click on the parent row, it needs to toggle the class on the <tbody> to toggle the correct class name
    4. Explore using the clickBubble: false bind handler to prevent the click event on Manage Duplicates also triggering a click on the parent row

45 mins

Exercise 10f: filtering & [x]

1. Clicking [x] on items in merge list should remove it from the list and update the state on the left hand side

2. The [x] icon on the merge list should not be visible if the that surgeon is set as the display name

3. Support filtering the data using Knockout computeds

45 mins

Exercise 11a: fit & finish

1. Table cells shift when you add something to the selected list. Correct this with table-layout: fixed

2. You get a flash of unused template while all the JS processes and loads the data. Fix this by hiding the templates with CSS and then displaying them via observables.