```
66    App.prototype.start = function (appConfig, retry) {
67        var self = this;
68
69        retry = retry || 0;
70
71        if (this.core.started === true) {
72            log.out('App is already started. Refusing to run start() routine again.');
73            return;
74        }
75
76        // set jQuery references
77        if (retry === 0) {
78            this.$loading = $('#mml-loading');
79            this.$app = $('#mml-desktop');
80            this.$body = $('body');
81
82            // attach the application shell template
83            this.$app.html(window[MML_APP_NAMESPACE].util.getTemplate('shell'));
84
85            // check to see if the loading curtain has been disabled for debug purposes
86            if (this.util.urlParams.disableLoadCurtain) {
87                this.$loading.remove();
88            }
89
90            // in case something goes drastically wrong during the remainder of the load routine, display messaging to user after 15 seconds
91            this.triggerLoadingInfoAfter(15000);
92
93            log.out('start() loading appConfig.json from:', appConfig);
94            this.radio.app.trigger('starting');
95        } else {
96            log.out('start() attempting to reload appConfig.json (retry #' + retry + ') from:', appConfig);
97        }
98
99        // ** load the appConfig and kick off all of the data routines
100       $.ajax({ url: appConfig })
101        .done(function (data) {
102           var section;
103
104           if (!window[MML_APP_NAMESPACE].util.isDef(data, ['api', 'base'], 'prop')) {
105               log.error('FATAL: Loaded appConfig.json but it did not contain an "api.base" section. Halting app initialization.');
106               return;
107           }
108
109           // store the raw config and notify early bird modules that the appConfig is available
110           self.json.config = data;
111           self.core.appConfig = appConfig; // save the path to the appConfig in case it has to be reloaded this session
```

Beginner JavaScript

by Matthew Stills

General Assembly Bootcamp

Part I: Introduction

Who Am I?

- Lead Developer, Turner (Broadcasting)
- Currently working in their sports division on the NCAA.com and March Madness Live web products
- Been developing websites for over 15 years
- The web technologies were much smaller when I started learning
  - HTML was much smaller than it is today
  - CSS had just been invented but not adopted
  - JavaScript rarely used for application-level work (ie. nothing like Gmail)
- There weren't as many resources available to learn from

Intro to Web Development

- The web employs three programming languages.

- These three languages work in harmony.

- Javascript can be used to alter parts of the other two languages. (We'll see this in action later!)

- Our focus today is on JavaScript, but we will encounter elements of the other languages.

This Workshop

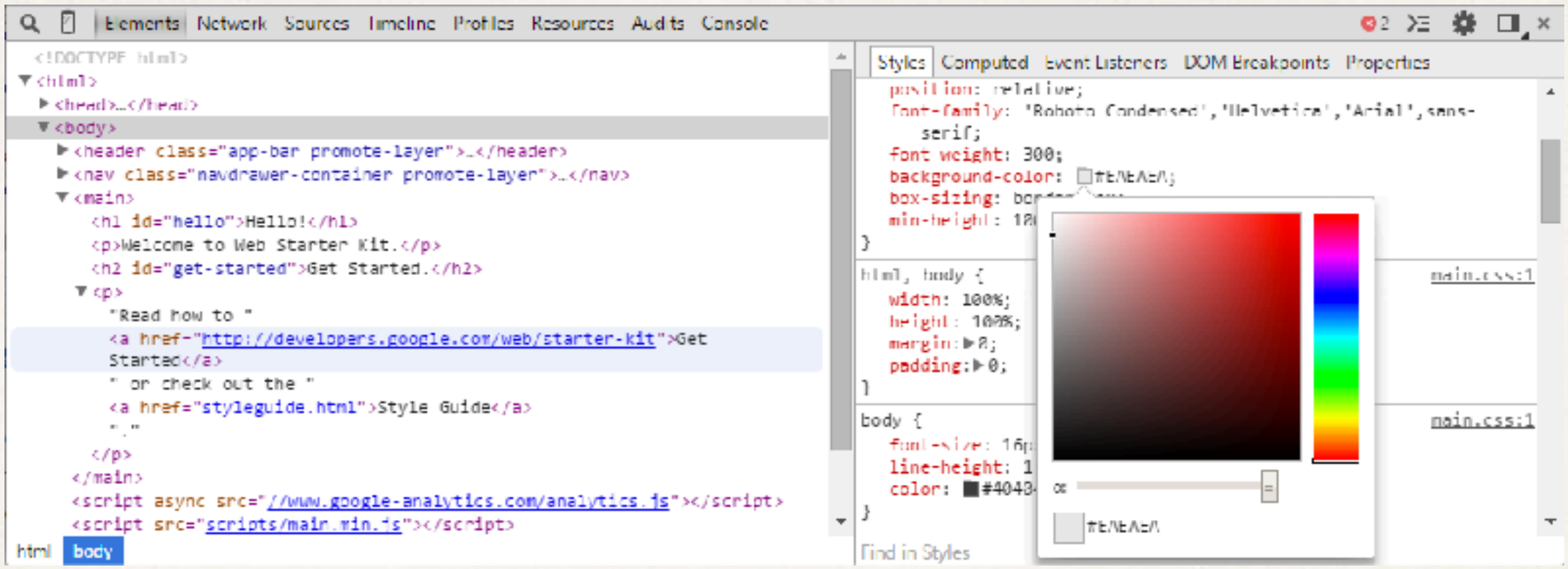
- Will cover:
  - Learning the fundamentals of *modern* in-browser JavaScript
  - Current version of core language only (ES5)
- Will not cover:
  - Any particular frameworks/libraries (eg. jQuery, Angular)
  - Cross-browser compatibility / JavaScript for legacy browsers
  - New, bleeding-edge JavaScript features only supported in the latest browsers (ES6)
  - JavaScript outside of the browser.

Web Frontend

- JavaScript is designed to work with the other two frontend languages.
- HTML provides the overall structure of a document via markup.
  - JavaScript can completely alter (add, remove, edit) the HTML.
- CSS provides the styling to that structure.
  - JavaScript can completely alter the CSS as well.
  - This is technically achieved the same was as HTML.
- JavaScript is a fully-fledged programming language that can dynamically modify the frontend according to the program logic.

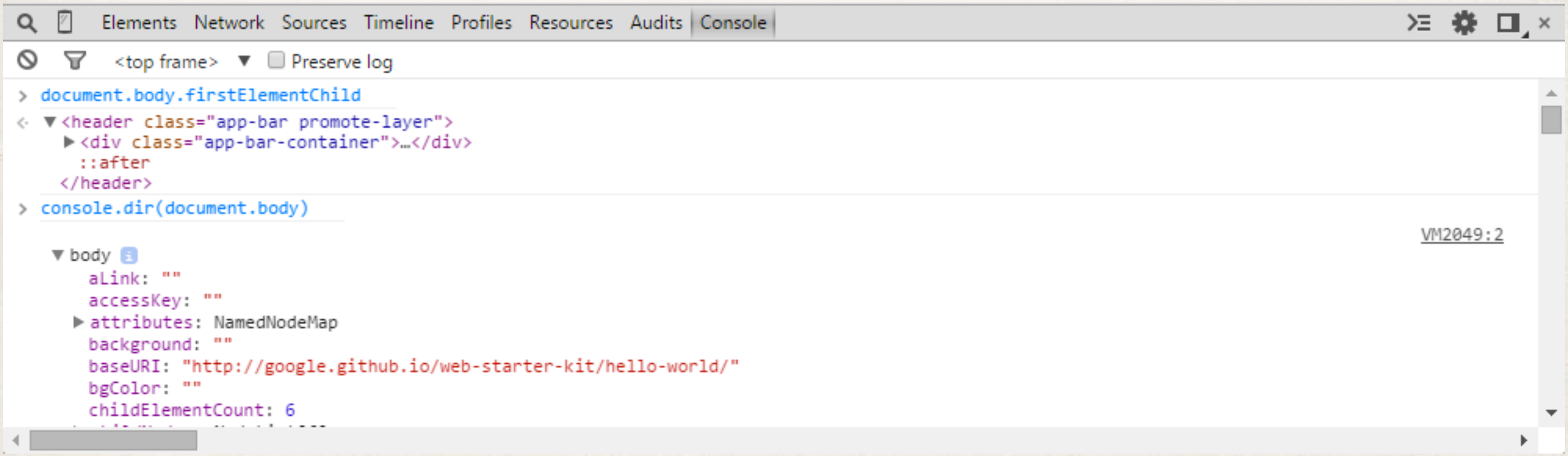
Our Toolkit

- Web Browser: Google Chrome
  - google.com/chrome
- Chrome Developer Tools (option/alt+shift+i)
  - Console
  - Elements
- Text editor for code: Sublime Text 3
  - sublimetext.com/3


Elements Network Sources Timeline Profiles Resources Audits Console
<!DOCTYPE html>
<html>
<head>…</head>
<body>
<header class="app-bar promote-layer">…</header>
<nav class="navdrawer-container promote-layer">…</nav>
<main>
<h1 id="hello">Hello!</h1>
Welcome to Web Starter Kit.
<h2 id="get-started">Get Started.</h2>

"Read how to "
Get Started
" or check out the "
Style Guide
"."

</main>
<script async src="//www.google-analytics.com/analytics.js"></script>
<script src="scripts/main.min.js"></script>
html body
Styles Computed Event Listeners DOM Breakpoints Properties
position: relative;
font-family: 'Roboto Condensed','Helvetica','Arial',sans-serif;
font-weight: 300;
background-color:
box-sizing:
min-height:
}
html, body {
width: 100%;
height: 100%;
margin: 0;
padding: 0;
}
main.css:1
body {
font-size: 16p
line-height: 1
color:
}
main.css:1
Find in Styles


Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
Preserve log
> document.body.firstElementChild
<header class="app-bar promote-layer">
…
::after
</header>
> console.dir(document.body)
VM2049:2
body
aLink: ""
accessKey: ""
attributes: NamedNodeMap
background: ""
baseURI: "http://google.github.io/web-starter-kit/hello-world/"
bgColor: ""
childElementCount: 6

Including JavaScript

- JavaScript is included using the HTML tag `<script>`
- You can include as much JavaScript in your HTML document as you like.
- It can be included in one of two ways.
- Inline:
- `<script>`
  - `// JS code is written right here!`
- `</script>`
- External:
- `<script src="http://www.site.com/index.js"></script>`

Part II: Programming Fundamentals

Syntax and Grammar

- Programming languages are comprised of rules just like spoken languages.

- It is easiest to point out examples of syntax and grammar related concepts as we encounter them as we progress throughout this lesson.

- Syntax will always be highlighted in blue code snippet examples.

Data Types

- Data comes in multiple types.
- These types are the building blocks of any language.
- Numbers: quantities!
    - Non-decimal integers
    - May be positive or negative
- Strings: sequences of characters or what we think of as "words/sentences"
    - Space is a character, all numbers and letters are characters.
    - Strings are always begin and end with a quotation mark.
    - Either double-quote ("") or single-quote ('') style is accepted.
- Boolean: true or false (with no quotation marks)
- Null: nothing (this is pretty abstract, we will see what the possible value of a data type that means "nothing" is as we progress through some exercises)

Operators

- Logical and mathematical operators let us work on data-types. Here are some of the most common:
- Equality (==)
- Greater than, less than (>, <)
- Greater than/equal to, less than/equal to (>=, <=)
- Math: (+, -, /, %)
  - Addition: +
  - Subtraction: -
  - Division: /
  - Modulo: % (to get the remainder of a division operation; useful for determining if a number data type is odd or even)
- Note: These operators will have a different effect depending on the data-type.
- Let's do a quick exercise to see what that all means!

Variables

- Variables are used to store your data which is expressed in one of the data-types we've already learned about.
- You can create a variable in two ways.
  - Without data: `var foo;`
  - With data: `var foo = 20;`
  - This is the only time you need to use the "var" keyword.
- Subsequent operations are done by just referencing the variable name:
  - `foo + 20`
  - `foo > 20`
  - `foo >= 20`
  - `foo % 2`
- The data-type of a variable may be checked using the "typeof" operator.

Conditionals

- Conditionals are the simplest of the control-flow logic features of JavaScript.
- The syntax for conditionals consists of one or more blocks.
- In JavaScript, blocks are bounded by "{" to open and "}" to close.
- The syntax for conditionals uses up to three keywords: if, and then optionally "else if" and "else" blocks.

```
if (1 > 2) {
    // do something if 1 is greater than 4
} else if (2 == 4) {
    // do something if 2 is greater than 4
} else {
    // do this if none of the other conditions evaluated to (bool)true
}
```

Loops: For

- A "for" loop will iterate over a snippet of code as many times as you specify.
- The syntax for specifying a "for" loop is:
  - `for (var i = 0; i < 10; i++) {`
  - `}`
  - Notice there are three sections:
    - The first section allows you to initialize one or more variables to assist with the loop iteration.
    - The second section is a conditional. The loop will continue to iterate and repeat the code within the { } block until this conditional evalutes to `(bool) false`.
    - The third section is an action to run every iteration. This is almost always an increment of the iterator variable set in the first section by 1 (`i++` is a concise way to perform this operation as we saw earlier).
- Let's try some "for" loops ourselves in the console sandbox!

Loops: While

- "while" loops are similar to "for" in that you set a condition and the loop continues until that condition evaluates to (bool) false.
- `var i = 0;`
- `while (i < 10) {`
  - `i++;`
  - `// other code here to be run until the while condition evaluates to (bool) true)`
- `}`
- This creates a "while" loop that behaves the exact same way as the "for" loop we saw last.
- While loops are useful in some cases but generally a "for" loop will suffice as it is safer (ie. you don't have to remember to manage the "while" condition inside of the block.

Functions

- Functions are crucial to any programming language.
- Functions allow you to create various units of code to be run at different times and allows for you to organize the different functionalities and routines that your JavaScript applications needs.
- JavaScript comes will many builtin functions that are very helpful (such as the Math functions that we'll use a little later).
- There are two function syntaxes to learn:
  - Function call
  - Function definition

Function Call

- Calling a function is simple.
- If a function takes no parameters, the syntax for calling it is simply:
  - `functionName();`
- If the function takes parameters, they are placed inside the parentheses:
  - `functionName(true, foo, 1);`
    - Example of passing a boolean, a variable called 'foo', and a number
- Some functions are part of an object or "namespace" (we'll learn about this more advanced datatype soon):
  - `Math.floor(1.11); // build in function for rounding a float decimal number down`
  - This is called "dot" notation. The word "Math" is considered the namespace, "floor" is the function name to call within that namespace. We'll learn more about this when we cover objects in the next section.

Function Definition

- Functions can be defined in one of two very similar ways.
- `function functionName() {`
    - `// function body`
- `}`
- This would define a function that takes no arguments.
- You may also assign functions to a variable:
- `var functionName = function () { // function body };`
- To define a function that accepts arguments when called (ie. the caller passes parameters as we saw in the last slide)
- var functionName = function (arg1, arg2) { / / arg1 and arg2 are variables pre-initialized with whatever value was passed in by the caller }
- So if I defined that function and then called it like so:
- `functionName(1, 'hello');`
- Then when that function gets executed, those variables will already be set:
- arg1 would equal 1, arg2 would equal 'hello'

Complex Data Types

- We looked at 3 simple data types already: string, number, and boolean.

- The next two are more advanced in that they "contain" multiple instances of the simple types.

- The "multiple instances" part is key.

- It enables a 3rd type of "loop" (the "for in loop") which we will learn about next.

Arrays

- The first complex data-type is an array.
- An array is an ordered collection of one or more simple data-types.
- Suppose we wanted to store the numbers 1-10 into variables.
- We could create ten variables separately but that isn't very useful.
- An array allows us to store 10 number data-type values into a single value.
- `var myFirstArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];`
- You may store strings, booleans, and even other arrays in an array.
- You may mix and types you want into an array; they do not all have to be numbers.
- An array can contain as many values as you desire.
- JavaScript provides built-in functions for manipulating arrays (inserting, removing, etc.)

Working with Arrays

- Get size of the array: `myArray.length`
- Add a value to end: `myArray.push(5);`
- Add a value to beginning: `myArray.unshift(5);`
- Remove a value from end: `myArray.pop();`
- Remove a value from beginning: `shift();`
- Iterating over an array: for loop
- Let's do a quick exercise!

Object (Literal)

- Objects are similar to arrays in that they hold multiple values and can be mixed an matched.
- Objects can also hold other objects (as well as arrays).
- Each value is an object is "named" with a key
    - Hence, an object can be considered a key / value store
    - Other languages have similar concepts: hash table, dictionary
- Objects have no sequence (ie. no necessary order)

```
var myObject = {
    id: 1,
    name: 'Matt',
    active: false // notice there is no "comma" here on the last entry
};
```

Object (Literal)

- The ability to hold other objects allows the object literal to act as a namespace.

- Allows us to form a hierarchy and organize related functions and values into the same place.

- Let's consider a "person" object and what it might need in an exercise!

Working with Objects

- Access the value of a key
  - by name: `myObject.key`
  - by name (that contains hyphens, spaces, or does not begin with letter):
    - `myObject['key-hyphenated']`
    - `myObject[123]`
    - `myObject['123four']`
  - by variable:
    - `var key = 'name';`
    - `myObject[key] // same as result a myObject['name']`
- See all keys: `Object.keys(myObject);`
- Removing a key: `delete myObject.name`

Looping over Objects

- `var currentKey;`
- `for (key in myObject) {`
  - `currentKey = myObject[key];`
- `}`
- Remember: There is not necessarily an order to the keys in objects. If you want to ensure you iterate over keys in a certain order, or if you wanted to pick specified keys, you can use a more advanced for loop technique.
- Let's practice both!

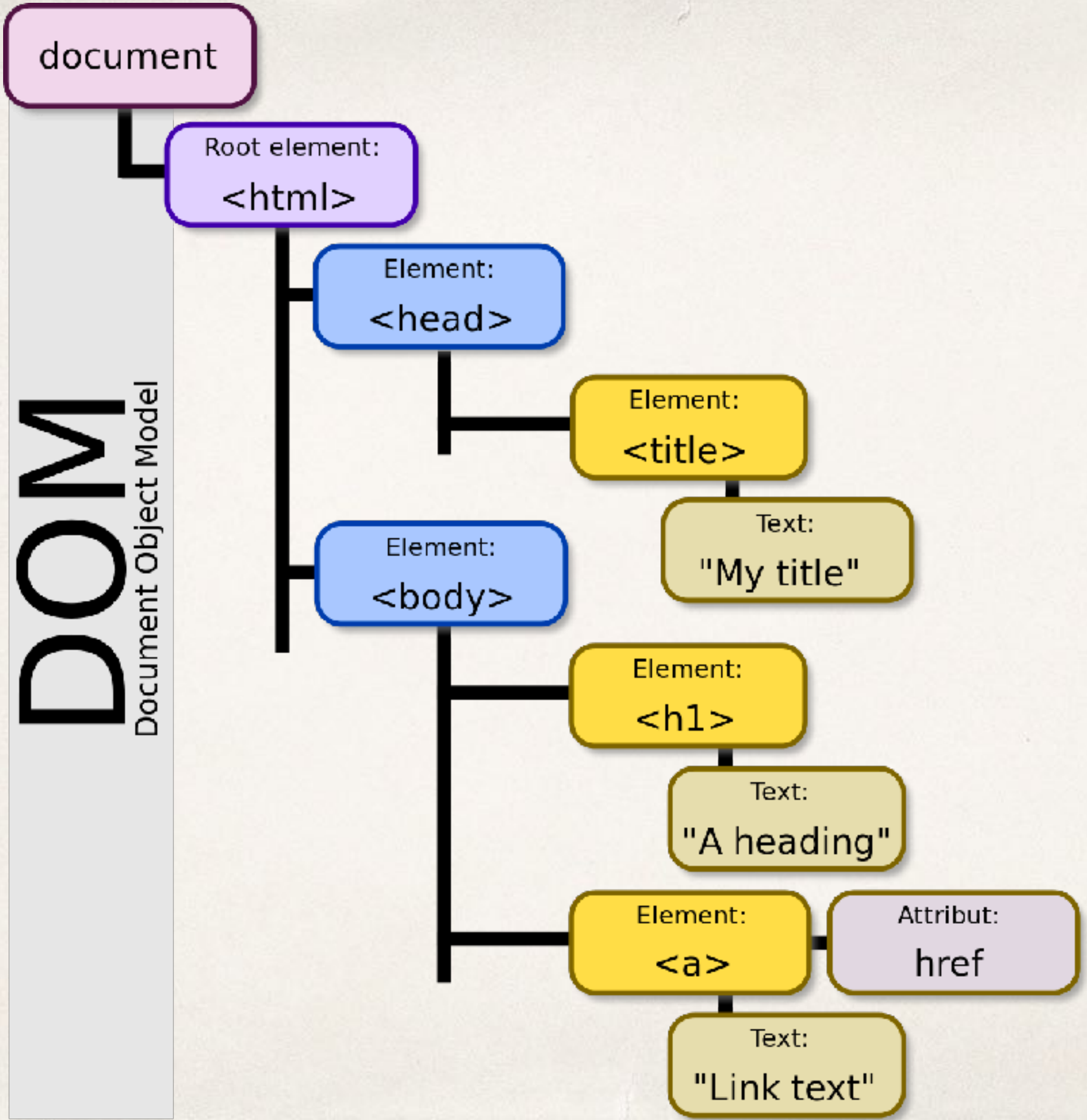
Part III: Document Object Model

What is it?

- Earlier we said JavaScript manipulates CSS and HTML via the same means.
- Both rely on the Document Object Model
- The DOM is an "API" (fancy term simply meaning a way for other programs to interface with itself)
- When an HTML document is parsed by the browser for rendering, it is also being parsed by the JavaScript engine into a tree that can be traversed in order to find elements to manipulate in some way.
- This will make more sense once we actually try a few uses! Let's do that now.

DOM

Document Object Model



DOM Functions

- In JavaScript, we interact with the DOM (ie. the HTML and CSS of a webpage that our JS is running on) via functions built-into the JS engine of every browser.
- This is where knowledge of HTML and CSS becomes important in JS.
- DOM functions are available under the "document" object/namespace:
  - `document.getElementById();`
  - `document.querySelectorAll();`
  - There are many others. The two above are the simplest to understand and use.
  - They are both selector functions.
- The easiest way to see how selectors work is to try them out!

Part IV: Debugging

Debugging

- Now that we know enough about JavaScript programming to be dangerous, we'll probably start to encounter bugs in code.
- A bug is simply anything that causes the program to not run the way it was intended to.
- Programmers of all skill-level have to debug their code.
- Therefore, learning HOW to debug effectively is very important!
- The tool we will be using today is the simplest means of "debugging" available.
  - Back-in-my-day note: When I started in web development, there was no means of debugging JavaScript code. It was all grueling trial-and-error.

Console Logging

- We've already seen the Console (in Chrome Dev Tools)
- We've used the console as our own "REPL" (READ-EVALUATE-PRINT-LOOP) for practicing some JavaScript concepts.
- Modern browsers also provide a "console" object/namespace with methods that you can use to allow JavaScript to "talk" back to the console.
- This might not sound like much but it enables incredibly powerful debugging through being able to inspect your code at any time, and report any information you want about the state of the code at that time.

Console Functions

- `console.log();`
- `console.info();`
- `console.warn()`
- `console.error();`
- These all print information to the console of your choosing.
- An error does not cause an error in program, it just sends that particular log to the "error" section of the browser's console.
- This allows for logs to be filtered by subject matter.

Part V: Async

Async vs. Sync

- So far we've been writing what is considered "synchronous" code.
- This simply means that the program is executed line by line, top to bottom, each line completing before the next line is executed.
- Synchronous code is considered to be "blocking" - the next line will not execute until the previous line is done executing.
    - If it is a function that takes a long time (such as an AJAX request that may take multiple seconds to get a response), it would block.
- Asynchronous code (which involves using asynchronous functions provided by JavaScript itself) does not necessarily execute top to bottom as we're about to see!
    - It does not block, an asynchronous function runs, and then any subsequent lines of code begin to execute immediately.
    - Many functions within JS are designed to be used asynchronously.
    - Asynchronous functions always involve providing a callback function.
    - The callback will be executed whenever the asynchronous function dictates
    - As always, it is easiest to see this in action.

Timers

- While a webpage is open, a JavaScript thread for it is running the entire time.
- If nothing is being executed, the thread will idle.
- Timing and interactions is a very important dimension of frontend development.
- The concept of timers helps introduce these concepts.

Timers, cont'd

- Timers allow you to schedule actions to occur at some point in the future.
- When setting a timer, you provide a function to be called after a certain amount of time passes.
- This is referred to as a callback function.
- JavaScript's standard unit of time is milliseconds.
- It's safe to assume that any function/API concerned with time is expecting you to represent time in ms
- Why?
  - Precision
  - LCD monitor can draw a new frame every 8-16ms
  - CPUs are fast enough to perform multiple operations even in that short span of time

Timers, cont'd

- JavaScript provides just two functions for setting timers.
- **setTimeout()**: waits N seconds and then executes the callback function once and only once.
- **setInterval()**: waits N seconds and then executes the function, waits another N seconds, executes that function again, and so forth until either the page is closed or some other part of the code explicitly turns the timer off (more on this later).

Timers, cont'd

- Timers are used to create any kind of JS-based animation.
- Remember that the unit of time in JavaScript is milliseconds?
- Within jQuery, JS animation is achieved by simply setting a timer with a short interval (say, every 200ms) to change a CSS property many times over.
- Timers are also used within dynamic applications to "poll" (ie. make continuous requests) a remote data source (this is AJAX, by the way!)
- Think of sitting on Twitter.com and after enough time passes, new tweets are available by clicking a link that suddenly appears at the top

Events

- Like timers, DOM events further demonstrate that every webpage's JavaScript is "living": it will run until the user closes their browser/tab!
- Oftentimes, JS code is intended to be executed at some point after the DOM has been created.
- Allows you to listen for events that occur throughout the lifespan of a webpage.
- The simplest example is a user clicking their mouse on something.
- When you tell JavaScript to listen for some type of event, you must also provide a function to be called when that event occurs.
- This is another example of a **callback** function!
- Let's do an exercise involving the simplest type of event: global listener.

Events

- JavaScript can listen to the entire page (by attaching event listeners to the "window" object as we just saw), or it can listen to individual elements within the DOM.
- Remember, each page has one global scope for all of the JavaScript on that page.
- It's unlikely that you "own" all of the JS running on a given page.
- Being able to listen to events only on certain elements that you are concerned with is one way of keeping your JS from interfering with the entire thread.
- In the case of a click listener, the click you are listening for most likely corresponds to a specific element.
- JavaScript's event system was designed to handle this.

Events

- Aside from clicks, JavaScript has a variety of other useful events that you can listen for.
- These include, but are not limited to:
  - movement of the mouse in or out of certain HTML elements
  - scrolling of the page
  - keystrokes
- Since the appearance of non-mouse and keyboard devices running web browsers, a variety of new event types have been added over the years.

Part VI: Scope

Variable Scope

- This is a bit more advanced.
- Important to know about nevertheless.
- Scope defines what variables are available.
- Depending on where you are (ie. inside a function vs. not), you scope may change.
- Each function has its own scope in JavaScript.
- This may seem confusing but it is a good thing!
- It allows for variable names to be re-used which allows for brevity.
- Imagine if you had to choose a new variable besides "i" for every "for" loop in your program!
- Let's do a simple scope exercise.

Part VII: Best Practices

Best Practices

- Use proper formatting and indentation!
- Write comments about any code that you've written that you think may be confusing to another programmer (or yourself a few months later when it's not fresh in your mind!)
- Err on the side of too many comments versus too few but do not explain obvious things:
  - `// storing important number for later`
  - `var myNumber = 100;`

Project Structure

- For small JavaScript (25-50 lines) you may not need to use an external *.js file.
  - Unless you have a specific reason to inline the JavaScript then it's always a good idea to use external *.js files!
- For anything longer or more significant, always use an external file.
- *.js files should range from 50-500 lines or so but not much more.
- When a file gets to be beyond that size, you should consider breaking it up into multiple files and including them all in the *.html via external `<script>` tags

Part VIII: More Resources To Continue Your Journey

Resources

- Codecademy
  - Intro to Programming (w/JavaScript)
    - https://www.codecademy.com/learn/javascript
      - Intro to JavaScript
        - https://www.codecademy.com/learn/learn-javascript
- MDN Reference
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Learning More About Chrome's Developer Tools
  - https://developer.chrome.com/devtools

Part IX: Parting Questions? Suggestions for how to improve this course?