Mobile App Development with React Native

Lectures

- Overview, JavaScript
- JavaScript, ES6
- React, JSX
- Components, Props, State, Style
- Components, Views, User Input
- Debugging
- Data
- Navigation
- Expo Components
- Redux
- Performance
- Shipping, Testing

Projects

- Project 0
- Project 1
- Project 2
- Final Project

Mobile App Development with React Native

Jordan Hayashi

Course Information

- Website
- Slack
- Staff email

Lectures

- Short break halfway
- Have a question? Interrupt me!
  - Concepts constantly build on each other, so it’s important to understand everything.
  - If something isn’t important to know, I’ll let you know
  - Staff will be monitoring Slack during lecture
- I love live examples!
  - Live coding has its risks. Let me know if you spot an error

Lecture 0: Overview, JavaScript

Jordan Hayashi

JavaScript is Interpreted

- Each browser has its own JavaScript engine, which either interprets the code, or uses some sort of lazy compilation
    - V8: Chrome and Node.js
    - SpiderMonkey: Firefox
    - JavaScriptCore: Safari
    - Chakra: Microsoft Edge/IE
- They each implement the ECMAScript standard, but may differ for anything not defined by the standard

Syntax

```
const firstName = "jordan";
const lastName = 'Hayashi';
const arr = ['teaching', 42, true, function() {
console.log('hi') }];

// hi I'm a comment
for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

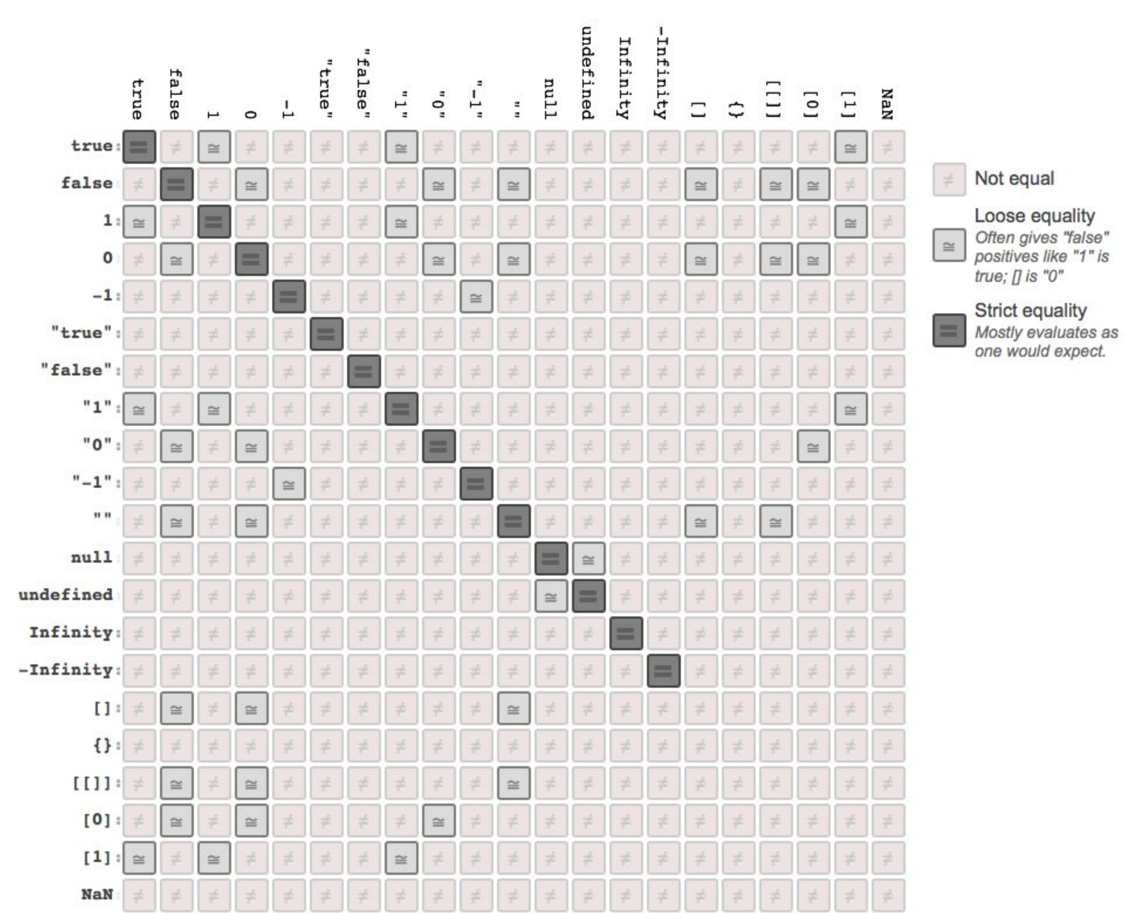
Types

- Dynamic typing
- Primitive types (no methods, immutable)
    - undefined
    - null
    - boolean
    - number
    - string
    - (symbol)
- Objects

Typecasting? Coercion.

- Explicit vs. Implicit coercion
  - `const x = 42;`
  - `const explicit = String(x);    // explicit === "42"`
  - `const implicit = x + "";       // implicit === "42"`

- `== vs. ===`
  - `== coerces the types`
  - `=== requires equivalent types`



	true	false	1	0	-1	"true"	"false"	"1"	"0"	"-1"	""	null	undefined	Infinity	-Infinity	[]	{}	[[]]	[0]	[1]	NaN
true	=	≠	≅	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	≠
false	≠	=	≠	≅	≠	≠	≠	≠	≅	≠	≅	≠	≠	≠	≠	≅	≠	≅	≅	≠	≠
1	≅	≠	=	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	≠
0	≠	≅	≠	=	≠	≠	≠	≠	≅	≠	≅	≠	≠	≠	≠	≅	≠	≅	≅	≠	≠
-1	≠	≠	≠	≠	=	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
"true"	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
"false"	≠	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
"1"	≅	≠	≅	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	≠
"0"	≠	≅	≠	≅	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	≠	≠
"-1"	≠	≠	≠	≠	≅	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
""	≠	≅	≠	≅	≠	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≅	≠	≅	≠	≠	≠
null	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	=	≅	≠	≠	≠	≠	≠	≠	≠	≠
undefined	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≅	=	≠	≠	≠	≠	≠	≠	≠	≠
Infinity	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠	≠
-Infinity	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	=	≠	≠	≠	≠	≠	≠
[]	≠	≅	≠	≅	≠	≠	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
{}	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
[[]]	≠	≅	≠	≅	≠	≠	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
[0]	≠	≅	≠	≅	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
[1]	≅	≠	≅	≠	≠	≠	≠	≅	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠
NaN	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠

≠ Not equal

≅ Loose equality
Often gives "false" positives like "1" is true; [] is "0"

= Strict equality
Mostly evaluates as one would expect.

https://github.com/dorey/JavaScript-Equality-Table

Coercion, cont.

- Which values are falsy?
  - `undefined`
  - `null`
  - `false`
  - `+0, -0, NaN`
  - `""`

- Which values are truthy?
  - `{}`
  - `[]`
  - Everything else

Objects, Arrays, Functions, Objects

- ^ did I put Objects twice?
- Nope, I put it 4 times.

- Everything else is an object
- Prototypal Inheritance (more on this later)

Primitives vs. Objects

- Primitives are immutable
- Objects are mutable and stored by reference

- Passing by reference vs. passing by value

Prototypal Inheritance

- Non-primitive types have a few properties/methods associated with them
  - Array.prototype.push()
  - String.prototype.toUpperCase()
- Each object stores a reference to its prototype
- Properties/methods defined most tightly to the instance have priority

Prototypal Inheritance

- Most primitive types have object wrappers
  - String()
  - Number()
  - Boolean()
  - Object()
  - (Symbol())

Prototypal Inheritance

- JS will automatically “box” (wrap) primitive values so you have access to methods

```
42.toString()        // Errors
const x = 42;
x.toString()         // "42"
x.__proto__          // [Number: 0]
x instanceof Number  // false
```

Prototypal Inheritance

- Why use reference to prototype?
- What’s the alternative?
- What’s the danger?

Scope

- Variable lifetime
  - Lexical scoping (var): from when they’re declared until when their function ends
  - Block scoping (const, let): until the next } is reached
- Hoisting
  - Function definitions are hoisted, but not lexically-scoped initializations
- But how/why?

The JavaScript Engine

- Before executing the code, the engine reads the entire file and will throw a syntax error if one is found
    - Any function definitions will be saved in memory
    - Variable initializations will not be run, but lexically-scoped variable names will be declared

The Global Object

- All variables and functions are actually parameters and methods on the global object
  - Browser global object is the `window` object
  - Node.js global object is the `global` object