JavaScript Foundations - Weekend Workshop

5 & 6 December 2015, Day One
WE Labs | Co-working, Long Beach CA

WiFi, source / slides

- WE LABS DOJO LB West
- Password: padnet2014
- Slides: https://github.com/Rockncoder/JSFoundations
- Please put your phones on vibrate

Troy Miles

- Troy Miles aka the RocknCoder
- Over 35 years of programming experience
- Wrote a few games for Interplay in the 80's and 90’s
- Speaker and writer on all things web & mobile
- rockncoder@gmail.com
- @therockncoder

Day One: The Fundamentals

- Intro to JavaScript
- Running in the browser
- Fundamentals
- Variables and data types
- Operators
- Program flow
- Objects
- Functions
- JSON
- Libraries
- Array methods
- Tips and Tricks

Day Two: Future Forward

- ES6 & TypeScript
- let/const
- Destructuring
- Arrow/lambda functions
- Maps and Sets
- Template Strings
- Classes
- Modules
- Promises
- Symbols
- JS Design Patterns
- Building Apps

“JavaScript is a high-level, dynamic, untyped, and interpreted programming language.”

–Wikipedia

Key points

- high-level - can used to build any kind of app
- dynamic - data can be changed on the fly
- untyped - variables can be use
- interpreted - isn’t compiled

JavaScript

- Created by Brendan Eich at Netscape in 1995
- Initial version developed in about 10 days
- Previously named Mocha and LiveScript
- "JavaScript" is a trademark of Oracle Corporation.
- Standardized by the Ecma first in 1997 as ECMA-262

Rankings of JS popularity

- 8th according to TIOBE
- 7th according to IEEE
- 7th according to the PYPL
- 2nd according to Mashable
- 1st according to RedMonk

Where is it used?

- Web browsers
- Servers via Node
- Databases via MongoDB
- Computer Games via the Unity game engine
- PDF files via Adobe's Acrobat & Reader

ECMAScript Versions

Version	Date
ES1	June 1997
ES2	June 1998
ES3	December 1999
ES4	DOA 2006
ES5	December 2009
ES6/ES2015	June 2015
ES2016	2016

Running in the browser

- index.html
- app.js

index.html

```
1     <!DOCTYPE html>
2     <html lang="en">
3     <head>
4       <meta charset="UTF-8">
5       <title>Hello</title>
6     </head>
7     <body>
8     <script src="app.js"></script>
9     </body>
10    </html>
```

`

```
1
2     // this is a JavaScript file
3
4     alert("Hello there.");
```

Fundamentals

Names

- Start with a letter, underscore, or dollar sign
- Followed by letters, underscores, dollar signs or numbers
- Used for statements, variables, parameters, property names, operators, and labels
- A name can't be a reserved words

Numbers

- Single 64-bit floating point, number type
- No separate integer type
- Exponents - 1e3 === 1000
- NaN - not a number
- Infinity is all values greater than 1.79769313486231570e+308

Strings

- Zero or more characters wrapped in either single or double quotes
- The backslash is the escape character
- Characters are 16 bit wide

Escaped characters

Character	Meaning
\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	tab
\v	vertical tab
\'	single quote (apostrophe)
\"	double quote
\\	backslash

Special symbols

Character	Meaning
\XXX	Latin-1 encoding using 3 octal digits
\xXX	Latin-1 encoding using 2 hexadecimal digits
\uXXXX	Unicode using 4 hexadecimal digits

Statements

- A compilation unit is composed of a set of executable statements.
- In web browsers, the <script> tag delivers a compilation unit
- It is compiled and immediately executed
- Execution begins at the top and works its way down

Kinds of statements

- expression statement
- disruptive statement
- try statement
- if statement
- switch statement
- while statement
- for statement
- do statement

Disruptive statements

- break statement
- return statement
- throw statement

if statement

- Changes the flow of execution based of the value of an expression
- If the expression is “truthy”, the *then* block is executed
- The else block is optional and executed when the expression is “falsy”

Falsy values

- false
- null
- undefined
- The empty string ‘’
- The number 0
- The number NaN

Truthy values

- All other values are truthy, including:
  - true
  - the string ‘false’
  - all objects, including empty ones

Scope

- Only functions create scope
- Blocks don’t create scope
- Variables should be defined at the top of function not in blocks

ES6 reserved words

- break case class catch const continue debugger
- default delete do else export extends finally for function
- if import in instanceof let new return super switch
- this throw try typeof var void while with yield

Upcoming reserved words

- await
- enum

Strict mode reserved words

- implements interface
- package private protected public static

ES1 - ES3 reserved words

- abstract boolean byte char double
- final float goto int long native
- short synchronized transient volatile

Comments

- Same as C++, C#, and Java
- // single line comment
- /*
  multi-line comment
  */

Variables

- var - declares a variable with optional initialization
- An uninitialized variable’s value is undefined
- undefined is both a type and a value
- var a; // type is undefined, value is undefined
- var b = “Happy"; // type is string, value is “Happy”

Data types

- string - “butter monkey” or ‘butter monkey’
- number - 64 bit floating point
- boolean - true or false
- undefined - both a type and a value
- null - both a type and a value
- object
- array

Type constructors

- DON’T USE THESE!
- Array()
- Object()
- Number(object)
- String(object)
- Boolean()

Conversion

- To convert from string to int, parseInt(string, radix)
- To convert from string to float, parseFloat(string)
- A shortcut to convert string to number, ‘+’

NaN, Not a number

- When there is an error while operating with numbers, NaN is generated
- It is toxic
- NaN != NaN
- To detect: IsNaN(value)

Operator precedence

Character	Meaning
. [] ()	field access, array indexing, function calls
++ — - ~ delete new typeof	unary operators, return, object creation
* / %	multiplication, division, modulo
+ - +	addition, subtraction, concatenation
<< >>	bit shifting
< <= > >= instanceof	comparison
!= === !==	equality, inequality
&	bitwise AND
^	bitwise XOR

Operator precedence (cont)

Character	Meaning
|	bitwise OR
&&	logical AND
||	logical OR
?:	conditional
=	assignment
,	multiple evaluation

ECMAScript 5 (ES5)

- Strict mode
- String
- Array
- JSON

Strict Mode

- 'use strict’; or "use strict;"
- First line of file or function
- Can break working code!!
- More stringent checking
- Enables ES5 features

Strict mode

- Variables must be declared
- Functions defined only at the top scope level
- “this” is undefined at the global level
- Throws error when you forget the 'new' keyword in constructor
- Can't use the same function parameter twice

Array functions

- .isArray()
- .every()
- .forEach()
- .indexOf()
- .lastIndexOf()
- .some()
- .map()
- .reduce()
- .filter()

JSON

- JSON strings must be double quote
- key/value pairs
- key is any valid JS string
- value is an valid JS data type
- .stringify() - converts object to JSON string
- .parse() - converts JSON string to object

JSON sample

```
1    {
2      "title": "U.S. Citizenship",
3      "tagLine": "Think you can pass this quiz of
basic knowledge of American history?",
4      "added": "2015-07-04T18:25:43.511Z",
5      "rating": 3,
6      "tags": [
7        "history",
8        "geography",
9        "United States"
10     ]
11   }
```

Performance tips

Minimize JS files

- Browser can load multiple files at a time
- But only one JS file at a time
- Concatenate multiple JS file into one
- Compress JS files
- Prefer JS at the bottom of the HTML file

Prefer local vars

- Variables in scope found quicker
- JS search local scope, then global
- with creates a new scope level ahead of local
- closures also create new scope level

Minimize object chains

- Property chains similar to var scoping
- Objects closer in the chain found quicker

Minimize work done in loops

- No speed difference between: for, while and do_while
- Avoid for_in
- Be wary of library based for_each (usually slow)

Avoid the DOM

- The DOM is REALLY Slow
- Avoid accessing it when possible
- Do work offline then update DOM

Groovy tips

Tip #1

Use protection

Use Protection

- The Browser is a very dirty environment
- Protect your code by wrapping it in a function

```
/* using protection */

(function (doc, win) {
   “use strict”;

  /* put all of your precious code here to keep it safe */
  /* extra bonus, parameters passed in become local, minor performance boost */

}(document, window));
```

Tip #2

debugger is your friend

debugger is your friend

- At times it can be difficult to set a breakpoint
- The debugger statement allows you to set a breakpoint any where you like

```
  app.post("/clientadmin", function (req, res) {
    var clientId = req.body.client;
    console.log("/clientadmin POST: " + JSON.stringify(req.body));
    if (clientId) {
      mongodb.getClientModel().findOne({_id: clientId }, function (err, client) {
        mongodb.getCampaignModel().find({clientId: clientId}, function (err, campaigns) {
          debugger;
          console.log("Campaigns: " + JSON.stringify(campaigns));
          /* set the client id */
          res.cookie('clientId', clientId);
          res.cookie('client', JSON.stringify(client));
          res.render("clientadmin.hbs", {client: client, campaigns: campaigns, user:
extractUser(res)});
        });
      });
    }
  });
```

Conditional

Tip #3

Always Use ===

Always Use ===

- Double equals (==) does automatic type conversion
- The results of this conversion is not logical or obvious
- Avoid this by using triple equals (===)
- There is no good reason to ever use ==
- Same goes for !=, use !== instead

Tip #4

Learn to love falsey

Learn to love falsey

- When coming from C# or Java it is tempting to use C-like conditionals
- JavaScript conditionals can be more succinct and performant

```
1  /* C-style conditional */
2  if (val != null && val.length > 0){
3     ...
4  }
5
6  /* JavaScript style conditional */
7  if (val) {
8     ...
9  }
10
```

Falsey

Type	Results
null	FALSE
undefined	FALSE
Number	if 0 or NaN, FALSE
String	if length = 0, FALSE
Object	TRUE

Conversions

Tip #5

Getting default value

Getting default value

- At times it is nice to have a default value
- JavaScript has a kind of default value operator

```
/* Conversions */
var defaultValue = defaultValue || 16;
var defaultString = inputValue || “Here is the default value”;

console.log(defaultValue);
```

Tip #6

Convert to boolean

Convert to boolean

- Double exclamation (!!) converts a value to its boolean representation (true/false)

```
/* convert to boolean */
var toBinary = !!null;
console.log(toBinary);
```

Tip #7

Convert string to number

Convert string to number

- The plus sign (+) before a numeric string converts it to a numeric value

```
/* convert to number */
var toNumber = +"42";
console.log(toNumber);
```

Tip #8

Convert value to string

Convert value to string

- Add an empty string (“”) to a value converts it to a string

```
var toString = 42 + "";
console.log(toString);
```

Design Patterns

ES6

Using ES6 support

- Most browser don’t support all of ES6 yet
- To check current support levels: https://kangax.github.io/compat-table/es6/

Transpiler

- A transpiler is a specific kind of compiler that translate from one language to another very similar language
- We use a transpile to convert es6 to es5
- We can also use on to convert from TypeScript to JavaScript
- Some transpilers are: Traceur, Babel, and TypeScript

Key features

- Template strings
- Arrow/lambda functions
- let
- const
- Array methods part 2
- Default values
- Destructuring
- Symbols
- Maps
- Sets
- Promises
- Classes
- Modules

Template strings

- a template string is indicated by the use of enclosing back ticks ` and ${}
- the ${} is a template and the value of the expression inside of it replaces it in the string
- template strings can run simple expressions like addition
- they can allow us to create multi-line strings

Template string example

```
1  var state = 'California';
2  var city = 'Long Beach';
3  console.info(`This weekend's workshop is in ${city}, ${state}.`);
```

Template string example

```
1  var cup_coffee = 4.5;
2  var cup_tea = 2.5;
3  console.info(`coffee: $${cup_coffee} + tea: $$
{cup_tea} = $${cup_coffee + cup_tea}.`);
```

Template string example

```
1  console.info(`This is line #1.
2  this is line #2.`);
```

Arrow/lambda functions

- arrow functions are syntactic sugar for the anonymous function you already know and love

Arrow function example

```
1  var a_func = function (num1, num2) {
2    return num1 + num2;
3  };
4  console.info(`Anonymous func: ${a_func(1, 2)}`);
5
6  var arrow_func = (num1, num2) => num1 + num2;
7  console.info(`Arrow func: ${arrow_func(3, 4)}`);
```

let

- let allows us to create a block scoped variables
- they live and die within their curly braces

let example

```
1  let val = 2;
2  console.info(`val = ${val}`);
3  {
4    let val = 59;
5    console.info(`val = ${val}`);
6  }
7  console.info(`val = ${val}`);
```

const

- const creates a variable that can't be changed
- does not apply to object properties or array elements

const example

```
1  const name = 'Troy';
2  console.info(`My name is ${name}`);
3  // the line below triggers a type error
4  name = ‘Miles';
```

Array.from()

- array.from has a lot of different uses
- creates an array from supplied length and arrow function
- make an array from a string

array.from() example

```
1  var instantArr = Array.from({length: 20}, (elem,
index) => index + 1);
2  console.info(instantArr);
3
4  // make an array from a string
5  var pirateArray = Array.from("happy pirates");
6  console.info(pirateArray);
```

for of

- for of lets us iterate over an array
- .entries() gives us the index and the element
- with destructuring we can use as variables
- unlike forEach, continue and break work
  - break stop the loop
  - continue stops the current iteration

for of example

```
1  // for of lets us iterate over an array
2  for (let elem of pirateArray) {
3    console.info(elem);
4  }
```

for of example

```
1  // getting the element and index
2  for(let [index, elem] of pirateArray.entries()) {
3    if (index === 5) {
4      break;
5    }
6    console.info(`${index}. ${elem}`);
7  }
```

Default values

default value example

```
1   var add = (x=1, y=2) => x + y;
2   console.info(add(10));
```

Destructuring

destructuring example

```
1  {
2     let obj = {first: 'Troy', last: 'Miles'};
3     let {first: f, last: l} = obj;
4     console.info(`My names is: ${f} ${l}`);
5  }
6
7  {
8     let [first, second] = pirateArray;
9     console.info(`1st = ${first}, 2nd = ${second}
`);
10 }
```

Symbols

Maps

Sets

Promises

TypeScript

Installing TypeScript

- npm install -g typescript
- npm update -g typescript
- tsc greeter.ts