Intro To JavaScript

History of JavaScript

It is always important to know the history of technologies. The technologies we use today aren't perfect, that is because we don't have perfect foresight. What we use today are a combination of what we thought would work at some point in the past, along with improvements that have been made along the way. Understanding that history helps us understand which parts of the technologies we are using make sense to use today, rather than mistakenly using something from the past that exists only for legacy reasons, not because it is recommended today.

- Source: http://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript
- Programming language (originally) targeted for scripting client side actions on a browser.
- Created over the period of 10 days in May of 1995 by Brendan Eich, working for Netscape.
- Original name was Mocha (by Marc Andreessen), then changed to LiveScript (November of 1995), then changed to JavaScript as a bit of a marketing gimmick to take advantage of the interest in Java (also released in 1995). It is only superficially related to Java with some of its syntax, it is otherwise pretty much nothing like Java.
- In 1996/1997 it was taken to ECMA (European Computer Manufactures Association) to work out a language standard.
    - Actual official name is ECMAScript.
    - JavaScript is an implementation of ECMAScript, ActionScript 3 is also an implementation of ECMAScript (with some extensions).
    - ECMAScript 5 is the baseline for JavaScript these days, with ECMAScript 6 supported by the latest and greatest browsers.
- Lot's of wrangling over what should happen with ECMAScript 4, which eventually led to its demise (a good thing),which eventually let to ECMAScript 3.1, and now renamed to ECMAScript 5.
- In June 2015 ECMAScript 6 is approved. Browser support of the language is good in Edge, Chome, and Firefox, but not Internet Explorer. Lot's of new features like constants, block-scoped functions/variables, modules, classes, iterators, enhanced object properties, promises, internationalization, and much more.

JavaScript Features

- Source: http://en.wikipedia.org/wiki/JavaScript
- **Imperative and Structured** (e.g., if statements, while loops, switch statements, etc).
- **Dynamic Typing**: Variables are just labels, they simply refer to a type & value somewhere in memory.
- **Object Oriented**: Almost everything is an object, with only a few primitives types. While it is object oriented, it does not have classes in the way that C++/C#/Java has classes. Well, except we now have 'classes' in ES6 (I don't use them).
- **Functional**: Functions are "first class" language members, they are in fact, objects themselves. Functions can have properties and methods!
- **Function Scope vs Block Scope**: Show a simple example; this will be detailed more later on.
- **Prototype-based Inheritance**: Each object refers to another prototype object. The source object can have methods/properties and the prototype also has methods/prototypes.
- **Runtime Environment**:
  - Typically runs in a web browser in the context of that environment and has access to a browser provided API, such as the DOM.
  - Can also run as standalone or as a server (e.g., Node.js using Google's V8 engine).
- **Garbage Collection**
- **Platform Independence** (especially with ECMAScript 5)
  - Across hardware (desktop, mobile, many CPUs)
  - Across OS (Windows, Linux, Mac, Android, etc).
- **Reflection**: Can discover object methods/properties at runtime
- **Integration with JSON**
- **Language Interoperability**: Can bridge with C++ using Google's V8 engine on the server.

Our First JavaScript Program

```
//
// Let's create a standard Hello World type program
let message = 'Hi Mom!';
console.log(message);
```

Anatomy of This Program

1. Comments: `//`
2. Variable: `let message`
3. String: `'Hi Mom'` alternatively `"Hi Mom"`
4. Statement: `let message = 'Hi Mom!';`
5. Environment API: `console.log(message);`

What We Don't See

1. Any kind of *main*. There is no such thing, JavaScript code is executed (or parsed) in the order it comes in the file.

   The above is a simple statement, but don't forget this is how JavaScript works, it is very important to understand this as you begin to write more complex programs that are composed of multiple files. ***JavaScript code is executed/parsed in the order it comes in the file***; hence, by order of code file.

Our Second JavaScript Program - Objects

```
let person = {
    firstname: 'John',
    lastname: 'Doe',
    age: 26
};
console.log('First Name: ' + person.firstname);
console.log('Last Name: ' + person['lastname']);
console.log('Age: ' + person.age);
```

Anatomy of This Program

1. object: let `person = {…};`
2. Defining object properties: `firstname: 'John'`
    - Properties are separated by commas
3. Property Access: `person.firstname`
4. Property Access: `person['lastname']`
5. Automatic conversion: `'Age: ' + person.age`

What We Don't See

1. The definition of an object is also the definition of the type. Unlike C+/C#/Java where a type is declared and then an instance made, in JavaScript, the instance is the type!

Variation #1 : Spaces in the property names

```
let person = {
    'first name': 'John',
    'last name': 'Doe',
    age: 26
};
console.log('First Name: ' + person['first name']);
console.log('Last Name: ' + person['last name']);
console.log('Age: ' + person.age);
```

Variation #2 : Nested object/properties

```
let person = {
    name: {
        first: 'John',
        last: 'Doe'
    },
    age: 26
};
console.log('First Name: ' + person.name.first);
console.log('Last Name: ' + person.name.last);
console.log('Age: ' + person.age);
```

Our Third JavaScript Program - Functions

```
function report(person) {
    console.log('First Name: ' + person.name.first);
    console.log('Last Name: ' + person.name.last);
    console.log('Age: ' + person.age);
}

let person = {
    name: {
        first: 'John',
        last: 'Doe'
    },
    age: 26
};
report(person);
person.name.first = 'Jane';
report(person);
```

Anatomy of This Program

1. Function: `function report(person) {…}`
   - Function declaration: `function`
   - Function name: `report`
   - Function parameters: `person`
2. Function body (or block): `{…}`
   - No semicolon after the block, only needed after statements

Notice the `person` parameter does not need a `let` indicator, or type of any kind! The type and properties are all worked out at run-time. (demonstrate this by making another object that has the same properties, but also some others, such as eye-color, and call report with that type).

Variation #1 : Function assigned to a variable

```
let report = function(person) {
    console.log('First Name: ' + person.name.first);
    console.log('Last Name: ' + person.name.last);
    console.log('Age: ' + person.age);
}

let person = {
    name: {
        first: 'John',
        last: 'Doe'
    },
    age: 26
};
report(person);
```