CS4220 Node.js & Vue.js

Cydney Auman
Albert Cervantes
CSULA

Intro Javascript

ECMAScript - ES5 and ES6

- The standard for JavaScript is ECMAScript.

- ES5 was released in 2009. That is still what a lot of developers are using today and what people know as modern JavaScript.

- ES6 was released in 2015 and is now becoming the new standard in development.

Core vs Client Side vs Server Side

Core JavaScript contains a core set of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects.

Client-side JavaScript extends the core language by supplying objects to control a browser and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.

Server-side JavaScript extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a variety of databases, provide continuity of information to and from the application, or perform file manipulations on a server.

Javascript Primitive Types

- **Number** – JavaScript does not define different types of numbers, like integers, short, long, floating-point etc. *They are always 64-bit Floating point.*
- **String** – In JavaScript strings can be created using single or double quotes.
- **Boolean** – true and false literals.
- **Undefined** – The value of "undefined" is assigned to all uninitialized variables, and is also returned when checking for object properties that do not exist.
- **Null** – Unlike undefined, null is often set to indicate that something has been declared *BUT* has been defined to be empty.

Installation and Setup

- Install Node.js (v8.9.4) - nodejs.org

- After install
  - in your terminal run the command: `node -v`
  - the output should be `v8.9.4`

- Install your prefered IDE
  - Sublime Text - https://www.sublimetext.com/3
  - VS Code - https://code.visualstudio.com/

Console.log and comments

Using **console.log()** outputs a message to the console.

Preceding a line of code with **//** will comment it out so that it will not execute.

```
console.log('hello')                // hello
console.log('hello' + ' world')      // hello world
console.log(1)                      // 1
console.log(1 + 1)                  // 2
console.log(false)                  // false
console.log(null)                   // null
console.log(undefined)              // undefined
```

Comparison Operators

The Equals Operator (==) (!=)

The == version of equality is quite liberal. Values may be considered equal even if they are different types, since the operator will force coercion of one or both operators into a single type (usually a number) before performing a comparison.

The Strict Equals Operator (===) (!==)

This one's easy. If the operands are of different types the answer is always false. If they are of the same type an intuitive equality test is applied: object identifiers must reference the same object, strings must contain identical character sets, other primitives must share the same value. NaN, null and undefined will never === another type.

Greater than (>)
Great than or equal (>=)
Less than (<)
Less than or equal (<=)

Pitfalls of Comparison

- Just because the value of a type is falsey ***does not*** mean that values of two different types are equal using the double equals. *(Ex, null and undefined)*

Declaration

Variables in standard JavaScript have no type attached, and any value can be stored in any variable. In ES5 variables were all declared using the keyword **var**.

ES6 introduced **const** and **let.**

Using **const** makes your variables a constant value. Variables defined using the keyword **const** will never be changeable.

Using **let**, is more similar to **var** in the sense that you can change the value assigned. However, where **let** and **var** differ is in relation to how they scope themselves.

Functions

JavaScript Functions are First-Class Objects

- They can be assigned to variables, array entries, and properties of other objects.
- They can be passed as arguments to functions.
- They can be returned as values from functions.
- They can possess properties that can be dynamically created and assigned.

Functions

JavaScript Functions are composed of four parts:

- The function keyword.
- An optional name that, if specified, must be a valid JavaScript identifier.
- A comma-separated list of parameter names enclosed in parentheses.
- The body of the function, as a series of JavaScript statements enclosed in braces.

```
function addTwo(n,m) {
  return n + m
}
```

Anonymous Functions

The function below is an **anonymous function** (a function without a name).
Functions stored in variables, do not need names. They are always invoked/called using the variable name.

```
const addThree = function(n){
  return n + 3
}
```

Arrow Functions

Arrow functions are functions defined with a new ES6 syntax that uses an “arrow” (=>).

An arrow function expression has a shorter syntax than a function expression and does not bind its own *this*, *arguments*, *super*, or *new.target*. Arrow functions are always anonymous.

```
const addThree = function(n) {
    return n + 3
}

// equivalent to:
const addThree = (n) => {
    return n + 3
}


// equivalent to:
const addThree = n => n + 3
```

Review & Prep

Review

- Slides
- Run Class Examples in Terminal
- Read Eloquent Javascript - Chapters 1 & 2

Preparation for Next Week

- Read Eloquent Javascript Chapters 3 & 4