CS454 Node.js & Angular.js

Cydney Auman
CSULA

Introduction to Javascript - Week 1

What is JavaScript?

JavaScript is a cross-platform, lightweight, interpreted, prototype-based object-oriented language with first-class functions. It is most known as the scripting language for web pages, but used in many non-browser environments as well such as Node.js.

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.

Java vs JavaScript

JavaScript	Java
Object-oriented. No distinction between types of objects. Inheritance is through the prototype mechanism, and properties and methods can be added to any object dynamically.	Class-based. Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically.
Variable data types are not declared (dynamic typing).	Variable data types must be declared (static typing).

Declaration

Variables in standard JavaScript have no type attached, and any value can be stored in any variable.

Variables are declared with a `var` statement, multiple variables can be declared at once.

```
var greeting = ‘hello world’;
var alpha, beta, gamma;
var alpha = 5, beta = 'hello’, gamma = 'world';
```

Declaration

JavaScript is dynamically typed - this means you don't need to specify the datatype of the variable when it is declared

In Java we would say:

```
int year = 2015;
String greeting = “hello world”;
```

BUT in JavaScript:

```
var year = 2015;
var greeting = ‘hello world’;
```

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.

Automatic Type Conversion

When an operator is applied to the “wrong” type of value, JavaScript will quietly convert that value to the type it wants, using a set of rules that often aren’t what you want or expect. This is called ***type coercion.***

JavaScript goes out of its way to accept almost any program you give it, even programs that do odd things.

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.

Pitfalls of Comparison

- Just because the value of a type is falsely does not mean that values of two different types are equal using the double equals.

Objects

There are a number of ways you can create objects in JavaScript.

1. You can directly instantiate an Object Object and then add your own properties and methods.

```
var car = new Object();
car.year = 2013;
car.make = 'Honda'
car.color = 'black';
```

Objects

2. You can use object literal notation to define your object.

```
var car = {
     year: 2013,
     make: 'Honda',
     color: 'black',
};
```

Arrays

In JavaScript, objects and arrays are handled almost identically, because arrays are a special kind of object. Arrays have a length property but objects do not.

1 Using new Array().

```
// An array with three elements
   var myArray = new Array(3);

   // Add some data
   myArray[0] = ‘Hello World’;
   myArray[1] = 88;
   myArray[2] = new Date(2015, 1, 1);
```

Arrays

2. Literal notation

```
   var myArray = [‘Hello World’, 88, new Date(2015, 1, 1)];
```

* If you assign a value to an element of an array whose index is greater than its length (for example, myArray[100] = "hello"), the length property is automatically increased to the new length.

* If you make the length property smaller, any element whose index is outside the length of the array is deleted.

Iterating Over Arrays

Arrays provide methods to iterate over and manipulate members. The following example shows how to obtain the properties of objects stored in an array.

```
var arr = ['hello', 'world', '!']

//using a for loop
for (i = 0; i < arr.length; i++) {
  console.log(arr[i])
}
// using the forEach method provided by Array
arr.forEach(function (word) {
  console.log(word)
})
```

Functions

You can create your own functions and use them where needed. A function definition consists of a function statement and a block of JavaScript statements.

```
// define a function with a name of addThree
function addThree(arg) {
  return arg + 3;
}

// define a function and set it to a variable
var addTwo =  function (arg) {
  return arg + 2;
}
```

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.

```
var addTwo =  function(arg) {
  return arg + 2;
}
```