305: JavaScript / Jquery

Dan Goldsmith

May 2018

Introduction

Introduction

- JavaScript Introduction
- Basic Syntax
- Functions
- A bit of DOM
- Objects

JavaScript Introduction

What is JS

- Language originally designed to allow interactivity in Web pages
  - Can modify HTML content
  - Can change HTML Styling
  - Can Validate data

JavaScript is not Java

- Designed for different use cases:
- Simple Objects VS Class based objects
    - Effects inheritance etc
- Dynamic VS static typing
- Ability to deal with low level system commands

Inserting Scripts into HTML pages

- We can place any number of scripts in a HTML document
- Can be either:
  - Inline (code is included in the HTML document)
  - External (code is loaded from another .js file)
- Can be placed anywhere in the document
  - It is a good idea to be consistent
  - Generally, the end of the file is used.

Inserting scripts into HTML

- Either Inline

```
<html>
  <body>
    <h1>Page Header</h1>
    ...
    <script> alert("HELLO WORLD!");</script>
  </body>
</html>
```

Inserting scripts into HTML

- Or Load from file

```
<html>
  <body>
    <h1>Page Header</h1>
    ...
    <script src="thescript.js"></script>
  </body>
</html>
```

Testing the Code:

- http://www.codeanywhere.com
- Using debugging tools in browser

Testing (Code anywhere)

- New Project
- HTML
- Create new file

Testing (Code Anywhere)

```
<html>
  <body>
    <h1>Page Header</h1>
    ...
    <script> alert("HELLO WORLD!");</script>
  </body>
</html>
```

- Preview

Testing the Code (Browser):

- Create the following and open in browser:

```
<html>
  <body>
    <h1>Page Header</h1>
    ...
    <script> alert("HELLO WORLD!");</script>
  </body>
</html>
```

JavaScript Basics

Comments

- Allow you to make notes for others reading the code.
- Vital to understanding your code
    - Trust Me! Its a very good habit to get into
- Are ignored by the interpreter

Comment Syntax

- Single Line

```
// This is a comment
```

- Multiline

```
// This is a
// Multiline Comment
```

A Better way of multiline

```
/*
This is a
Multiline comment
*/
```

Variables

- Core building block of code
- Used to hold data
- Called Variables as their value can differ

Basic JavaScript Types:

- **Number** 42, 3.14
- **String** “Hello”
- **Boolean** true, false
- **Null** null

Complex JavaScript types:

- **Array** [“foo”, “bar”, “baz”]
- **Dictionary / Object** {“key1”: “value1”, “key2”: “value2”}
- **Dates**
- **Functions** function()
- **undefined**

Complex JavaScript types:

- Note that arrays and dictionaries can hold different types of object
  - Or even nested objects
  - Basis of JSON

```
[[1,"foo"],[2,"Bar"]]

[{id:1,name:"Dan"},
 {id:2,name:"James"}]
```

Arrays

- Represent Lists of items
    - 0 Indexed
    - *length* tells us the array size

Creating Arrays (1):

```
var a = new Array()
a[0] = "foo"
a[1] = "bar"
a[2] = "baz"
a.length //returns 3
```

Creating arrays (2):

```
var a = ["foo","bar","baz"]
a.length // returns 3
```

Getting / setting elements in the array

- Use the Index (remember it starts at 0)

```
var a = ["foo", "bar", "baz"]
a[1] = "bleh"
// A then becomes
["foo", "bleh", "baz"]
```

Array Methods

```
a.toString()   //Convert to string
a.join(sep)    //Covnvrt to string separated by "sep"
a.pop()        //Good for Stacks
a.push(item)   //Insert item(s) at end of array
a.reverse()
a.slice(start, end)  //take a subset of the array
a.sort()
```

Declaring Variables:

- Defined using either the **var** or **let** keywords

```
var a
var name="Dan"
```

- If the variable has no value assigned its type is *undefined*

Defining variables

```
var name="Dan"
```

- **var** Keyword defining a variable
- **name** Name of variable
- **value** Initial value of variable

Variable Scope

- If defined using **var** then the variable is visible to the entire function.
- If defined using **let** then visible only to the statement

Variable Scope: Var VS Let

```
var name="Dave"

if (name == "Dave"){
   let answer = "Foo"
   }
else {
   var answer = "Bar"
   }

alert(answer)
```

Basic operators: Assignment VS Equality

- Single Equals is **Assignment**
  - x = 5 (set the value of X to 5)
- Double Equals is **Equality** (Return true if X = Y)
  - x == 5 (true)

Basic Operators: Mathematical

- Standard Maths applies
    - Addition (+)
    - Subtraction (-)
    - Division (/)
    - Multiplication (*)
    - Modulo (%)

Basic Operators: Increment / Decrements

- ++ Add 1 to the value
  - 5++ (6)
- – Subtract 1 from value
  - 5– (4)
- += Increase by a given value)
  - 5+=3 (8)
- -= (Decrease by a given value)
  - 5-=3 (2)

Basic operators: String Concatenation

- We also use + for string concatenation. So:

```
"Foo" + "Bar" == "FooBar"
```

Basic Operators: Comparison

- Standard functions
- == Equal to
- **!=** Not Equals
- < Less than
- > Greater Than
- <= Less than or Equal to
- >= Greater than or Equal to

Comparison: Strict versions.

- There are also strict versions of equality
  - **===** , **!==**
- Check the Variable type is the same.
  - Consider Boolean's, 0 and False are logically the same

```
0 == false (true)
0 === false (false)
```

Basic Operators: Logical

- **&&** Logical AND
- – Logical OR

```
1=1 && 2=2 (true)
1=1 && 2=1 (false)
1=1 || 2=1 (true)
```

JS Basics: Control Structures

Selection: IF

- Make something happen *IF* a given condition is True

```
var number=42
var output
if (number == 42){
   output = "Meaning of Life"
}
```

Selection: If Else

- We can chain statements together

```
var grade = 65
var result
if (grade < 40){
   result = "fail"
}
else if (grace < 70){
   result = "pass"
}
else {
   result = "distinction"
}
```

Selection: Switch

- Can be used for multiple branches based on numbers or strings

```
var name = "Dan"
switch(name) {
   case "Dan":
      job = "Lecturer"
      break
   case "James":
      job = "Senior Lecturer"
      break
   default:
      job = "student"
}
```

Selection: Switch Ranges

- Either *fallthrough* to the first statement

```
int day = 0;  //Numeric
switch(day){
   case 1: //Monday
   case 2: //Tuesday
   case 3:
   case 4:
   case 5:
      result = "Weekday"
      break
   case 6:
   case 0:
      result = "Weekend"
   default:
      result = "NA"
```

Iteration: For

- Loop over a Known number of objects
    - Same as C / Java etc.

```
for (var 1 = 0; i < 5; i++){
  //Do something
)
```

For Loops and Arrays

- Either a standard for loop

```
var array = ["foo", "bar", "baz"]
for (var i=0; i< array.length; i++){
   //do something with array[i]
   }
```

- Or For In syntax

```
for (var item in array){
  //do something with *item*
}
```

Iteration: While

- Loop until a given condition is true

```
var x = 0
while (x < 5){
   //do something
   x += 1 //REMEMEBR TO UPDATE CONDITION
   }
```

Iteration: Do While

- Checks the condition after the loop is run.
  - Good if we need to ensure the loop runs at least once

```
var input;
do {
   input = get_input()
} while input != 5
```

Your Turn

- In either CodeAnywhere or a Standalone file
  - Create some basic Variables (int, String, List etc)
  - Get familiar with iteration
  - You can use either console.log or Alerts to show data.

Functions

Functions

- Good programming practice
  - Allow us to break code into logical “tasks”
  - Avoids code duplication
  - Promotes code reuse

What is a good candidate for a function?

- Anything that may be used several times in the code
- A logical “Block” of code that performs a specific task

Function Definition

- Should be familiar to any programmers
- Core component in understanding JS

```
function add(x, y) {
    var total = x + y
    return total
}

var result = add(5,10)
//Result should now be 15
```

Breaking down the function definition

- Function Keyword **function**
- Name **add**
- Parameters **(x,y)**
- Function Body between **{..}**

```
function add(x, y) {
...
}
```

Function Names

- Used to call the function
- Choose something sensible
  - ie **add calculateArea**
  - Not *A B*
- Some Conventions for naming (these may differ between companies)
  - **functionName**
  - **function_name**

Function Parameters

- Can take 0 or more named parameters
- These are variables that are *passed* to the function for use in body
    - Again choose sensible names.

Function Body

- The block of code that makes up the function
    - Should be an Independent block of logic
    - Can have as many lines as you want (be sensible)
    - Can have its own variables

Return Value

- Value returned by the function
    - Can be any valid JS variable
- NOTE: return will immediately exit the function
- If no return value is used JS will return undefined.

Examining Functions:

- Is this a good function? What does it do?

```
function maths(foo, bar){
   var total = foo + bar
   return total
   }
```

Examining Functions 2:

```
function area(radius){
   var result = 3.14 * (r * r)
   return result
```

Types of Function: Named Functions

- This is the function type we are familiar with

```
function add(x, y) {
    var total = x + y
    return total
}
```

Types of Function: Anonymous Functions

- We assign a function to a variable. This means we can change the function at run-time

```
var  add = function(x, y) {
    var total = x + y
    return total
}
```

Calling Functions

- Call with Brackets () containing the required parameters
- add(1,2) // returns 3
- Passes control of the program to the function
  - Parameters are passed across
  - Also two “Hidden” parameters, (this, arguments)

Function Scope

- JS uses function scope
- All variables defined inside a function are **visible** within **THAT** function
- Variables defined outside a function are **NOT VISIBLE** inside the function

The DOM

DOM:

- Document Object Model
  - Abstraction of the tags used in an HTML page
  - We can use this to access and modify parts of our HTML page

The DOM:

- A Tree representation of the HTML within your web page
- The root element of the DOM is the *document*
- Each HTML tag is a Node
    - Each Node my have children
    - Each node will have a parent

Functions for locating items in the DOM

- getElementById()
- getElementByName()
- getElementByClass()

HTML Tags: Id’s

```html
<div id="TheDiv">
 ... Some Content ...
</div>
```

Modifing the content of Tags

We can do this in JavaScript

```
<script>
  var div = document.getElementById('theDiv')
  div.innerHTML = "Modified By Javascript"
</script>
```

What about Creating a new Element

```
window.onload = function() {
  // Create some elements
  heading = document.createElement("H1")
  headingText = document.createTextNode("The Title");
  heading.apendChild(headingText)
  document.body.appendChild(heading)
```

Your Turn

Your Turn

- Get a copy of the first Lab from GitHub (download / Clone)
- Work through the First example

A Note on the DOM

- There are a couple of concepts here we haven't yet looked at although they will be self explanatory.
- window.onload
    - Event triggered after the page has finished loading.
- document.getElementById
    - Return the DOM object with a given Id

Objects in JavaScript

Classless JavaScript

- Until recently JS had no class
    - True Class was introduced in ECMA6
- Classes were “faked” through the use of Objects

Objects

- Can be thought of as simple collections of name:value pairs
    - The Dictionaries in above example
    - Can also contain functions

Name Value Pairs

- Name is a string
  - Quotes are optional if it would be a valid JS variable name
- Value can be any JS value
  - String, Integer, List etc
  - Includes other Objects

Objects

```
var employee = {
   firstName: "Dan",
   secondName: "Goldsmith",
   department: "Hacking",
   dateOfBirth: new Date("12 Aug 1980")
   }
```

Creating Simple Objects

- Use an object literal

```
var emptyObject = {}

var employee = {
   firstName: "Dan",
   secondName: "Goldsmith",
  }
```

Creating Simple Objects: Constructor Function

```
function Employee(firstName, lastName){
   this.firstName = firstName
   this.lastName = lastName
}

var Dan = new Employee("Dan", "Goldsmith")
```

Nested Objects

```
var nested = {
    name:"Dan",
    subjects: {
        "245": "Ethical Hacking"
        "307": "Pervasive Computing"
        }
    }
```

Getting / Setting Object Values

- Either use brackets

```
employee["name"]  //Returns Dan
```

- Or Dotted Syntax

```
employee.name //Also Rturns Dan
```

Functions as Object properties

- Remember that an object can contain *ANY* JavaScript object
- This includes Functions
    - Gives Class like functionality
- When the function is called, it occurs within the object

Functions as Object Parameters

```
var countingObject = {
   value: 0,
   increment: function(inc) {
      this.value += inc
  }
}
```

Functions as Object parameters

```
countingObject.value  //0

//Increase vale by 5
countingObject.increment(5)
countingObject.value //returns 5

countingObject.increment(1)
countingObject.value //returns 6
```

Your Turn

Your Turn

- Get a copy of the Second Lab from GitHub (download / Clone)
- Work through the example